

Methoden zur Unterstützung bei der Entwicklung plattformübergreifender Benutzerschnittstellen



Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

vorgelegt von

Dipl.-Psych. Kai Richter
geboren in Stuttgart

Referenten der Arbeit:
Prof. Dr.-Ing. J. L. Encarnação
Prof. H. Wandke

Tag der Einreichung: 16.02.2007
Tag der mündlichen Prüfung: 04.05.2007

D17
Darmstädter Dissertationen 2007

Für Sonja & Theo

Danksagung

Die vorliegende Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter im Zentrum für Graphische Datenverarbeitung in Darmstadt.

Mein besonderer Dank gilt Herrn Professor Dr. José Luis Encarnação für die Förderung dieses interessanten Themas und für die Betreuung meiner Arbeit. Insbesondere möchte ich mich dafür bedanken, dass er es mir als Diplom-Psychologen ermöglicht hat, am Fachbereich Informatik der Technischen Universität Darmstadt zu promovieren. Weiterhin gilt mein besonderer Dank Herrn Professor Wandke, der meine Arbeit als Korreferent betreute.

Ich bin all jenen Personen zu Dank verpflichtet, die durch ihre Beteiligung an den empirischen Untersuchungen diese Arbeit erst möglich gemacht haben. Ihre Geduld und Kooperationsbereitschaft, wie auch die vielen wichtigen Anregungen und Verbesserungsvorschläge waren mir eine große Hilfe.

Ich danke Professor James D. Foley für den Hinweis auf die Arbeiten von Cooper und Shepard. Gabriele Schmid von der Universität der Künste Berlin danke ich für ihre Anregungen zur Ästhetik und die Hinweise zu den Werken von Arnheim. Ich möchte Anthony Jameson vom Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) danken für seine wertvollen Hinweise auf die Arbeiten von Nielsen und Anderson. Ich danke Antti Oulasvirta, Helsinki Institute of Technology, für die anregenden Diskussionen. Ich danke Jeff Nichols, Carnegie Mellon University, Krzysztof Gajos, University of Washington und Ahmed Seffah, Concordia University Montreal, für die Zusammenarbeit bei der Planung des Konsistenz-Workshops auf der CHI 2006. Ich danke allen Teilnehmern und Betreuern des Doctoral Consortiums auf der CHI Konferenz 2005 in Portland, sowie allen Kollegen, die mir in vielen Diskussionen die halfen die Konzepte dieser Arbeit zu entwickeln.

Mein Dank gilt den Partnern im Forschungsprojekt EMBASSI, Olaf Perlick und Frank Reins vom Forschungsinstitut Technologie-Behindertenhilfe, Volmarstein, Peter Albert und Kolja Sträter von der Handy Tech Elektronik GmbH, Horb, Professor Wandke und Marita Enge von der Humboldt Universität, Berlin, Stefan Lindner und Dr. Lothar Seveke vom Ingenieurbüro Seveke, Dresden.

Danken möchte ich Dirk Balfanz und Rainer Malkewitz für ihre Unterstützung. Ich danke Matthias Finke für die wertvollen inhaltlichen Diskussionen und für seine großartige menschliche Unterstützung. Mein Dank gilt weiterhin Volker Roth, Eric Blechschmitt, Michael Hellenschmidt, Matthias Grimm und all den anderen Kollegen aus dem ZGDV und Fraunhofer Institut für Graphische Datenverarbeitung für die Möglichkeit des Austauschs und für die gegenseitige Unterstützung. Vielen Dank auch an die wissenschaftlichen Hilfskräfte, Praktikanten und Diplomanden Alexander Oros, Tao Wei, Nina Valkanova, Jürgen Effenberger, Marc Machenheimer, Timo Kopp, Frank Herold und Simone Lanz, die mich bei der Umsetzung meiner Ideen unterstützten.

Ohne meine Familie wäre diese Arbeit nicht möglich gewesen. Ich danke meinen Eltern, meiner Großmutter wie auch meinen Schwiegereltern dafür, dass sie stets für mich da waren.

Danke Sonja.

Inhaltsverzeichnis

Zusammenfassung	XI
1 Einführung	1
1.1 Motivation	1
1.1.1 Das Zeitalter des Personal Computers	2
1.1.2 Jenseits des Personal Computers	6
1.1.3 Zusammenfassung	10
1.2 Lösungsansatz	11
1.3 Abgrenzung und Einordnung der Arbeit	12
1.4 Gliederung	13
2 Analyse der Aufgabenstellung und Ableitung von Anforderungen	15
2.1 Analyse der Aufgabenstellung	15
2.2 Empirische Untersuchung der Benutzeranforderungen	16
2.2.1 Einleitung	16
2.2.2 Erfassung des mentalen Modells	17
2.2.3 Aufgabenanalyse	19
2.2.4 Fehleranalyse	21
2.2.5 Methode	25
2.2.6 Benutzerbelastung und Zufriedenheit	27
2.2.7 Auswertung der Videodaten	28
2.2.8 Auswertung der Untersuchungen zum mentalen Modell	38
2.2.9 Diskussion	43
2.3 Anforderungen an die Benutzbarkeit	45
2.3.1 Benutzerwissen und Konsistenz	45
2.3.2 Benutzbarkeit	45
2.3.3 Barrierefreiheit	48
2.3.4 Zusammenfassung	53
2.4 Anforderungen an den Entwicklungsprozess	53
2.4.1 Nutzen	54
2.4.2 Kompatibilität	55
2.4.3 Komplexität	58
2.4.4 Testbarkeit	61
2.4.5 Sichtbarkeit	61
2.4.6 Zusammenfassung	62
2.5 Technische Anforderungen	62
2.5.1 Plattformunabhängigkeit	63
2.5.2 Skalierbarkeit	63
2.5.3 Dynamische Erweiterbarkeit	64
2.5.4 Standardkonformität	64
2.5.5 Zusammenfassung	65
2.6 Zusammenfassung	65
3 Grundlagen	67
3.1 Wissenstransfer	67
3.2 Mentale Repräsentation	70
3.3 Konsistenz	76
3.3.1 Dimensionen der Konsistenz	79
4 Methoden der Entwicklung von Benutzerschnittstellen	83

4.1	Windows-System	83
4.2	User Interface Toolkits	83
4.3	User-Interface Management Systeme (UIMS)	89
4.3.1	Spezifikationssprachen	90
4.3.2	Spezifikation durch Modelle	91
4.3.3	Existierende modellbasierte Ansätze	95
4.4	Messung der Konsistenz von Benutzerschnittstellen	105
4.4.1	Metriken	106
4.4.2	Messung der Konsistenz	114
4.5	Zusammenfassung und Bewertung	115
5	Ein Konzept plattformübergreifender Entwicklung	119
5.1	Benutzerzentrierte Entwicklung	120
5.2	Referenzmodell der plattformübergreifenden Entwicklung	123
5.2.1	Sequenzielle Anpassung	126
5.2.2	Geordnete Entwicklungspfade	130
5.2.3	Das Transformationsparadigma	133
5.3	Transformationsstrategien	138
5.3.1	Graphische Anpassungsmechanismen	138
5.3.2	Erhaltende Transformationen	140
5.3.3	Neuanordnung	141
5.3.4	Reduktion	143
5.3.5	Vereinfachung	143
5.4	Plattformübergreifende Konsistenz	145
5.4.1	Konsistenz plattformübergreifender Benutzerschnittstellen	145
5.4.2	Referenzmodell der Konsistenz multipler Benutzerschnittstellen	149
5.4.3	Ein transformationales Konsistenzmaß	151
5.4.4	Berechnung der transformationalen Konsistenz	155
5.4.5	Exemplarische Konsistenzberechnung	161
5.5	Zusammenfassung	165
6	Realisierung der Entwicklungskonzepte	167
6.1	Eine Darstellungsplattform	167
6.1.1	Übersicht	168
6.1.2	Einsatzszenarien	173
6.1.3	Anwendungsintegration	177
6.1.4	Datenmodell	182
6.1.5	Abstrakte Präsentation	184
6.1.6	Standard Präsentation	189
6.1.7	Design Präsentation	191
6.1.8	Zusammenfassung	193
6.2	Design-Unterstützung	194
6.2.1	Spezifische Anforderungen	194
6.2.2	Unterstützung konsistenten Designs	195
6.2.3	Unterstützung plattformübergreifender Konsistenz	204
6.3	Zusammenfassung	218
7	Systemvalidierung	219
7.1	Validierung des Darstellungssystems	219
7.1.1	Das Leitprojekt EMBASSI	220
7.1.2	Benutzerstudie	225
7.2	Validierung der Transformationsstrategien	230
7.2.1	Überprüfung der Transformationshypothese	230
7.2.2	Anwendung einer regelbasierten Transformation	237
7.3	Validierung des Konsistenzmaßes	245
7.3.1	Methode	246
7.3.2	Ergebnisse	247

7.3.3	Diskussion	253
7.4	Zusammenfassung	253
8	Zusammenfassung und Ausblick	255
8.1	Zusammenfassung	255
8.1.1	Inhalte	255
8.1.2	Ziele	258
8.2	Wissenschaftlicher Beitrag	259
8.3	Ausblick	260
	Literaturverzeichnis	263

Abbildungsverzeichnis

1.1	Ein Xerox Star 8010 Computer von 1981.	2
1.2	Der Xerox Star 8010 Desktop von 1981.	3
1.3	Der Microsoft Windows XP Desktop von 2001.	5
1.4	Die Vision der Ambient Intelligence.	7
1.5	Zugänglichkeit von Informationstechnik im öffentlichen Raum	10
1.6	Kontext- und geräteübergreifender Anwendungsentwicklung.	11
2.1	Plattformübergreifende vs. anwendungsübergreifende Entwicklung.	16
2.2	Ansicht des CardSword Programms zur Visualisierung von Begriffsklustern.	19
2.3	Use Case Diagramm der Aufgaben in der Benutzerstudie	22
2.4	Objekthierarchie der Anwendung	23
2.5	Fehlertaxonomie von Reason [Rea90]	24
2.6	Fehler und Ursprung des zugrundeliegenden Wissens.	26
2.7	Dialog zur Erstellung eines Kontakts.	32
2.8	Dialog zur Erstellung eines Kontakts auf dem PC.	32
2.9	Adresseingabemaske auf PC	33
2.10	Das Menü des Adressbuches auf dem PocketPC.	34
2.11	Erstellung neuer Email und Posteingang	35
2.12	Eingabe eines neuen Termins direkt in die Tagesansicht des Kalenders.	37
2.13	Eingabe eines neuen Termins über die Eingabemaske.	37
2.14	Beispiel für direkte Externalisierung, Anwendungsstruktur	39
2.15	Beispiel für direkte Externalisierung, Homogenität	40
2.16	Beispiel für direkte Externalisierung, Übersichtlichkeit	41
2.17	Produktionen für verschiedene Typen von Dialogen	46
2.18	Kategorien der ISO Standards nach [AKSA03].	47
2.19	Ein Schichtenmodell der Benutzbarkeit nach [Wel01].	48
2.20	Öffentliche Informationssysteme	49
2.21	Das Evaluation-Execution Modell von Norman [Nor88].	52
2.22	Das Assistenzkonzept nach Wandke.	57
2.23	Entwicklung plattformübergreifender Oberflächen.	59
2.24	Konsistenz vs. plattformspezifische Optimierung	60
3.1	Klassifikationsschema externaler und mentaler Repräsentationen nach [EK93].	70
3.2	Übersicht der Modelle.	73
3.3	Beispiel für eine inkonsistente Gestaltung.	76
3.4	Alternative Mentale Repräsentationen der Menüstruktur.	77
4.1	Abstraktionsebenen.	84
4.2	GUIDE Entwicklungsumgebung der Firma 3Soft.	85
4.3	Microsoft Visual Studio .Net: Desktop.	87
4.4	Microsoft Visual Studio .Net: Pocket PC	87
4.5	Logische Komponenten eines UIMS.	89
4.6	Entwicklung von Benutzerschnittstellen in einem UIMS nach [Göt95].	91
4.7	Die Architektur von XForms.	97
4.8	Architektur des V2 Standards.	98
4.9	V2 Beispielimplementierung.	99
4.10	CAMELEON Referenz Framework nach [CCB ⁺ 03].	100
4.11	Reifikationsprozess nach [TCC04].	101
4.12	USIXML Werkzeuge nach [Van05].	101
4.13	Abbildung der Aufgaben auf Objekte der Domäne.	102

4.14	Gestaltung des AUI mittels des IDEALXML Werkzeuges.	103
4.15	GRAFXML Editor.	103
5.1	Der Usability Engineering Lifecycle von Mayhew [May99].	122
5.2	Referenzmodell plattformübergreifender Entwicklung.	124
5.3	Bearbeitungsrollen und damit verbundene Arbeitsbereiche	126
5.4	Der Darstellungsprozess	127
5.5	Ableitung des Abstrakten User Interfaces.	128
5.6	Sequenzielle Anpassung	129
5.7	Einordnung des Referenzmodells.	131
5.8	Primäre und Sekundäre Entwicklungspfade.	132
5.9	Anwendung des Prinzips der geordneten Pfade.	133
5.10	Perspektiven des plattformübergreifenden Systems.	134
5.11	Modellbildung beim Entwickler und beim Benutzer.	134
5.12	Zwei Entwicklungsstrategien.	137
5.13	Erhaltende Transformationen.	140
5.14	Transformationen der Neuordnung	142
5.15	Strategien der Reduktion.	143
5.16	Vereinfachungsstrategien.	144
5.17	Ein Referenzmodell der Konsistenz multipler Anwendungen	149
5.18	Anpassung eines User Interface an eine Plattform.	152
5.19	Der Verlauf des Konsistenzmaßes für verschiedenen Varianzen.	159
5.20	Abhängigkeiten der Konsistenz	160
5.21	Schematische Darstellung der Transformation der Form.	162
5.22	Schematische Darstellung der Transformation der Position.	164
6.1	Realisierungsansätze für UIMS.	168
6.2	ARCH Modell.	169
6.3	Architektur des Darstellungssystems.	171
6.4	Lokaler Einsatz auf einer Plattform	174
6.5	Lokaler Einsatz auf einer Plattform als multimodale Plattform	174
6.6	Einsatz über ein Netzwerk.	175
6.7	Browser-basierte XFORMS Anwendung.	176
6.8	Anwendungsintegration über zwei Abstraktionsebenen.	177
6.9	Komponenten der Programmierschnittstellen.	178
6.10	Auszug aus der Klassenhierarchie der WinXAPI.	179
6.11	Hauptkomponenten der Schnittstelle.	179
6.12	Daten im Modell.	182
6.13	Rolle des Modells.	182
6.14	Verarbeitung von XML Ereignissen.	183
6.15	Die abstrakte Präsentation	184
6.16	Darstellung eines abstrakten XFORMS Elements	185
6.17	Auswirkung der Appearance Eigenschaft auf die Darstellung	186
6.18	Beispiel für eine Systempalette zur konsistenten Formatierung.	186
6.19	Vererbungshierarchie der abstrakten Darstellungselemente.	187
6.20	Zugriff auf die konkreten Darstellungselemente.	187
6.21	Die Komponenten der Standard Präsentation	189
6.22	Beispielhafte Darstellung einer einfachen Anwendung auf drei Plattformen.	189
6.23	Beispielhafte Darstellung eines Reduktionsprozesses.	190
6.24	Die Design Präsentation.	191
6.25	Beispiel für Design Unterstützung.	196
6.26	Beispiel für vertikale Konsistenzlinien.	197
6.27	Nutzung von Konsistenzlinien.	197
6.28	Dynamische Templates.	198
6.29	Formatvorlage in Microsoft Word.	199
6.30	Formatvorlage bei der UI-Entwicklung.	200
6.31	Beispiel für die Verwaltung von Textressourcen	202

6.32	Beispiel für Wortergänzung	202
6.33	Beispiel für eine Korrekturhilfe in Microsoft Word	203
6.34	Beispiel für eine kontextsensitive Konsistenzprüfung	203
6.35	Beispiel eines Entwicklungswerkzeuges	205
6.36	Unterstützung plattformübergreifenden Designs.	206
6.37	Anzeige der konsistenten Position.	207
6.38	Rückmeldung über die Verbesserung der Konsistenz.	208
6.39	Beispiel für eine Transformationsanwendung	209
6.40	Die Optimierung des Transformationsprozesses	210
6.41	Beispiel für die Transformation einer Element-Gruppe	211
6.42	Transformation zwischen Plattformen.	212
6.43	Beispiel eines HCI-Pattern von Welie.com	213
6.44	Patternunterstützte Übersetzung von Benutzerschnittstellen.	214
7.1	Eine visuell eingeschränkte Person.	221
7.2	Multimodales Dialogkonzept.	222
7.3	Übersicht über die Architektur des Systems.	223
7.4	Multimodale Ein- und Ausgabe.	224
7.5	Eine gelähmte Person bei der Nutzung des Systems.	226
7.6	Subjektive Bewertung des Aufwandes.	228
7.7	Bewertung der Benutzbarkeit	229
7.8	Mentalen Rotation.	230
7.9	Dialoge in den verschiedenen Orientierungen.	233
7.10	Ansicht der Versuchsanwendung.	234
7.11	Verteilung der Reaktionszeiten.	235
7.12	Verteilung der bereinigten Reaktionszeiten über die Orientierungen.	236
7.13	Verteilung der bereinigten Pausenzeiten für die drei gedrehten Dialoge.	237
7.14	Verteilung der bereinigten Reaktionszeiten über alle drei Eingaben.	238
7.15	Die scrollbare und die umschaltbare Toolbar.	241
7.16	Ein Ausschnitt der Microsoft Word Anwendung.	241
7.17	Die umschaltbare Toolbar auf einem Pocket PC.	242
7.18	Die Microsoft PocketWord Anwendung.	242
7.19	Akzeptanzwerte des SUS.	243
7.20	Differenzen der Bearbeitungszeiten pro Aufgabe.	244
7.21	Beispiel für die Fahrzeugdaten.	247
7.22	Beispiel für die konsistente Abbildung	248
7.23	Mittel konsistente Abbildung	249
7.24	Hoch inkonsistente Abbildung	250
7.25	Konsistente gespiegelte Abbildung	251
7.26	Mittlere Gesamt-Bearbeitungszeiten.	252
7.27	Veränderung der subjektiven Benutzbarkeitseinschätzung.	253

Tabellenverzeichnis

1.1	Charakteristika des Personal-Computers	4
1.2	Bandbreite der Interaktions-Charakteristika von Mobilgeräten.	8
2.1	Ablauf der Studie	27
2.2	Klassifikationsmatrix der Benutzerfehler	29
2.3	Die Handlungsebenen von Norman als Klassifikation von Ansätzen zur Unterstützung	51
3.1	Kelloggs Schema zur Definition von Konsistenz.	80
4.1	Übersicht verwendeter Modelle	94
4.2	Modellierung in XI ML	105
4.3	Einordnung der Metriken in Kelloggs Schema zur Definition von Konsistenz.	113
4.4	Bewertung der wichtigsten Ansätze bezüglich der Anforderungen.	117
5.1	Klassifikation der Transformationsstrategien	139
5.2	Häufigkeitsverteilung der Abbildung der Formen	163
5.3	Berechnung der Varianzen und Konsistenz der Translation.	164
7.1	Vollständigkeit und Genauigkeit.	227
7.2	Zeitersparnis und Reduzierung des Aufwandes.	228
7.3	Design der Studie.	232
7.4	Ergebnis der schrittweisen linearen Regressionsanalyse	234
7.5	Standard Eigenschaften der beiden Plattformen.	239
7.6	Bearbeitungszeiten.	244
7.7	Testbedingungen	246

Zusammenfassung

In der vorliegenden Arbeit werden Methoden zur Unterstützung der Entwicklung plattformübergreifender Benutzerschnittstellen vorgestellt. Ziel der Arbeit ist es, Entwicklern von Anwendungen für verschiedene Endgeräte (wie Mobiltelefone und Desktop Computer) Möglichkeiten an die Hand zu geben, mittels derer Anwendungsoberflächen einfach und für den Benutzer nachvollziehbar angepasst werden können. Weiterhin ermöglicht diese Flexibilisierung der Darstellung eine bessere Anpassung auf die Bedürfnisse behinderter Nutzer im Sinne des *Universalen Designs*. Kosten durch mehrfachen Entwicklungs- und Wartungsaufwand, Benutzungsprobleme durch inkonsistente Softwareversionen und Ausgrenzung durch unzugängliche Oberflächen sollen so vermieden werden.

Die in dieser Arbeit vorgestellten Konzepte wurden anhand einer benutzerzentrierten Vorgehensweise entwickelt. Softwaretechnische Lösungen werden hierbei stets im Kontext der kognitions- und wahrnehmungspsychologischen Erkenntnisse der Interaktion zwischen Mensch und Computer, insbesondere der Modelle der Repräsentation von Anwendungen (Mentale Modelle, Analoge Repräsentation) wie auch der Handlungsunterstützung (Assistenz, Anpassung), betrachtet und validiert. Um die Relevanz und Validität der Konzepte und Lösungen sicherzustellen, werden während der Analyse der Anforderungen, wie auch während der Evaluierung der Ergebnisse empirische Untersuchungen angestellt.

Die Ergebnisse dieser Arbeit unterstützen die zentrale Bedeutung der Konsistenz plattformübergreifender Anwendungen. Dies wird bereits empirisch aus der Analyse der Anforderungen abgeleitet. Es wird gezeigt, dass eine konsistente Gestaltung von Oberflächen für verschiedene Plattformen deren Benutzbarkeit erhöht. Methoden zur Unterstützung des Entwicklers bei der Gestaltung konsistenter Anwendungen stellen daher ein wesentlicher Beitrag zur Verbesserung der Benutzbarkeit und zur Vereinfachung des Entwicklungsprozesses dar.

Die wichtigsten Beiträge dieser Arbeit beziehen sich auf die Modellierung des plattformübergreifenden Entwicklungsprozesses. Wesentliche Ergebnisse sind die Konzeption und Umsetzung eines *User Interfaces Management System* für multiple Darstellungen, ein kognitionspsychologisch motiviertes *Vorgehensmodell zur transformationalen Anpassung graphischer Benutzerschnittstellen* und die Definition einer *Metrik zur Beschreibung der Konsistenz verschiedener Darstellungen*, welche beispielhaft in verschiedenen entwicklungsunterstützenden Methoden umgesetzt wurde.

Die Umsetzung des Darstellungssystems, sowie dessen erfolgreicher Einsatz in einer Anwendung zur Unterstützung behinderter Nutzer bei der Bedienung von öffentlichen Terminalsystemen wird im letzten Teil der Arbeit beschrieben, ebenso wie die empirische Validierung der Grundannahmen über die Rolle analogischer Repräsentationen beim Wechsel zwischen Plattformen. In zwei Studien wird gezeigt, dass das entwickelte Konsistenzmaß in der Lage ist, die Benutzbarkeit einer Anwendung auf verschiedenen Plattformen vorherzusagen.

Die Ergebnisse dieser Arbeit bieten damit die Möglichkeit, den Entwicklungsprozess von Benutzerschnittstellen für plattformübergreifende Anwendungen qualitativ, bezüglich der Benutzbarkeit der Produkte, aber auch quantitativ, durch eine Beschleunigung und Vereinfachung des Prozesses selbst zu unterstützen.

1 Einführung

Shouldn't software be designed so that users could run the same calendar program on a palm-sized device, a laptop, and a wall sized display?

— Ben Shneiderman, *Leonardo's Laptop*, 2003

Der zunehmende Einsatz von Informationstechnik in den verschiedensten Bereichen unseres Lebens spiegelt sich in der Vielgestaltigkeit der informationstechnischen Endgeräte mit denen wir uns heute konfrontiert sehen, wieder. Der klassische Personal-Computer mit Bildschirm, Maus und Tastatur steht heute vor allem in Konkurrenz mit verschiedenen mobilen Endgeräten, die in Funktion und Leistungsfähigkeit in absehbarer Zeit mit dem Personal-Computer gleichziehen werden. Aber auch im öffentlichen Leben, zu Hause oder unterwegs trifft man immer häufiger auf Computer in Form von eKiosk Systemen, Unterhaltungselektronik oder Navigationstechnik. Es ist sehr wahrscheinlich, dass dieser Trend sich auch in Zukunft verstärkt fortsetzen wird.

Die parallele Existenz von Computern mit stark unterschiedlicher Hardware und Software Konfiguration, man spricht auch von verschiedenen Plattformen, führt zu der Notwendigkeit, Anwendungen, für jede dieser Konfigurationen parallel zu entwickeln und zu pflegen. Insbesondere die unterschiedlichen Anforderungen an Benutzungsoberfläche, d. h. die meist graphischen und akustischen Ausgaben, mittels derer das System mit dem Benutzer kommuniziert, stellen eine große Herausforderung bei der Entwicklung dar. Die Nutzung derselben Anwendung auf verschiedenen Plattformen führt dazu, dass der Lernaufwand bzw. der Aufwand beim Wechsel um so größer ist, je gravierender die spezifischen Unterschiede in der Umsetzung der Anwendung sind.

In der vorliegenden Arbeit wird ein Lösungsansatz für das Problem der plattformübergreifenden Anwendungen, und speziell deren Benutzungsoberflächen, vorgestellt und evaluiert.

Zunächst wird ein System zur plattformunabhängigen Darstellung multimodaler Benutzungsoberflächen entwickelt. Dieses System reduziert den Entwicklungsaufwand für solche Systeme indem es die Erzeugung von verschiedenen plattformspezifischen Benutzungsoberflächen aus einer abstrakten plattformunabhängigen Definition erlaubt. Es wird gezeigt, wie die Benutzbarkeit von Informationstechnik vor allem auch für behinderte Benutzer dadurch verbessert werden kann, dass ihm stets die individuell optimal angepasste Konfiguration von Interaktionsmitteln zur Verfügung steht.

Werkzeuge, welche die plattformübergreifende Entwicklung von Benutzungsoberflächen unterstützen sollen, müssen Vorschriften über die Abbildung von Oberflächen auf die Spezifika verschiedener Plattformen implementieren. Eine systematische Ordnung verschiedener Abbildungsmöglichkeiten graphischer Oberflächen und deren Bewertung dient als Grundlage der plattformübergreifenden Darstellung.

Um die Qualität der Abbildung bewerten zu können und um den Aufwand des Benutzers beim Wechsel zwischen den Plattformen durch eine hohe Konsistenz der Abbildung reduzieren zu können, wird im zweiten Teil dieser Arbeit ein Index entwickelt, welcher die relevanten Faktoren zur Bestimmung der Ähnlichkeit zwischen Anwendungsoberflächen zusammenfasst. Die Möglichkeit einer programmatischen Umsetzung dieses Ähnlichkeitsmaßes wird diskutiert.

Der folgende Abschnitt beschreibt die Motivation dieser Arbeit. Es wird versucht nachzuzeichnen, weswegen die Entwicklung plattformübergreifender Benutzungsoberflächen und Anwendungen notwendig ist und wie Informationstechnik im Kontext der Universal Accessibility durch einen solchen Ansatz ihrer gesellschaftlichen Verantwortung gerecht werden kann.

1.1 Motivation

[...] usability methods are much more durable than the average computer technology. Usability concerns human capabilities and behavior. If you wanted to estimate how much the average IQ has

increased from 1989 to 2002, the answer would probably be less than one percent. Small improvements do happen as a result of better nutrition and such, but basically people are quite the same today as they were 13 years ago.

— Jakob Nielsen, Vorwort zur zweiten Auflage von *Coordinating User Interfaces for Consistency*, 2002

Die Bedeutung der Benutzungsoberfläche als Schnittstelle zwischen Mensch und Computer ist wohl unumstritten. Im Zweifel entscheidet die Benutzbarkeit und Attraktivität des User Interface (UI) über Erfolg oder Misserfolg eines Produktes. Trotz der Notwendigkeit durch Alleinstellungsmerkmale Aufmerksamkeit auf sich zu ziehen und Kunden an sich zu binden, hat sich die Software-Industrie in den letzten zwanzig Jahren, vor allem nach dem Aufkommen graphischer fensterbasierter Oberflächen, dazu durchgerungen, gemeinsame Konzepte zur Darstellung der Anwendungsoberflächen einzusetzen. Über verschiedene Hersteller und Systeme hinweg, wurden Anwendungen vereinheitlicht und Oberflächenelemente (sog. *Widgets*), wie z.B. Schaltflächen oder Listen, in gleicher Weise wiederverwendet. Dies erleichterte es dem Benutzer, sich in neue Anwendungen einzuarbeiten. Entwickler und Hersteller von Anwendungen konnten durch die konsistente Verwendung von vorgefertigten Konzepten Entwicklungsaufwand sparen und Entwicklungszeiten verkürzen.

Mit dem Aufkommen der mobilen Informationstechnik hat sich dieses Bild wieder verändert, die erreichte Homogenität wurde durch neue Geräte und Interaktionskonzepte (Plattformen) zerstört. Neue Entwicklungsmethoden zur Entwicklung von plattformübergreifenden Anwendungen sind notwendig geworden. Zugleich bietet die Verfügbarkeit von mobiler Informationstechnik eine große Chance hochwertige Unterstützung für Menschen mit eingeschränkter motorischer oder kognitiver Leistungsfähigkeit anzubieten. Computer bieten die Mittel, diesen Menschen dabei zu helfen ihre Defizite auszugleichen. Hierzu ist der plattformübergreifende und multimodale Zugang zu Informationstechnik notwendig.

Im folgenden Abschnitt wird dargelegt, welche Entwicklung zu der heutigen Situation der heterogenen und inkonsistenten Bedienkonzepte geführt hat und wie diese mit Hilfe eines Konzeptes zu plattformübergreifenden Bereitstellung und Entwicklung von Benutzungsoberflächen behoben werden kann.

1.1.1 Das Zeitalter des Personal Computers



Abbildung 1.1: Ein Xerox Star 8010 Computer von 1981 mit Prozessor, Bildschirm, Tastatur und Maus. Der Xerox Star war der erste Personal-Computer mit fensterbasierter graphischer Benutzerschnittstelle und Mauseuerung. (Quelle: <http://www.aci.com/mwichary/articles>)

Mit der Vorstellung des Xerox 8010 Star Information System—dem ersten Personal-Computer mit graphischer fensterbasierter Benutzerschnittstelle (GUI), Mauseuerung und Tastatur—im April 1981, begann eine neue Ära in der Mensch-Maschine-Interaktion, die den Umgang mit Informationstechnik bis heute prägt [SIKH82].

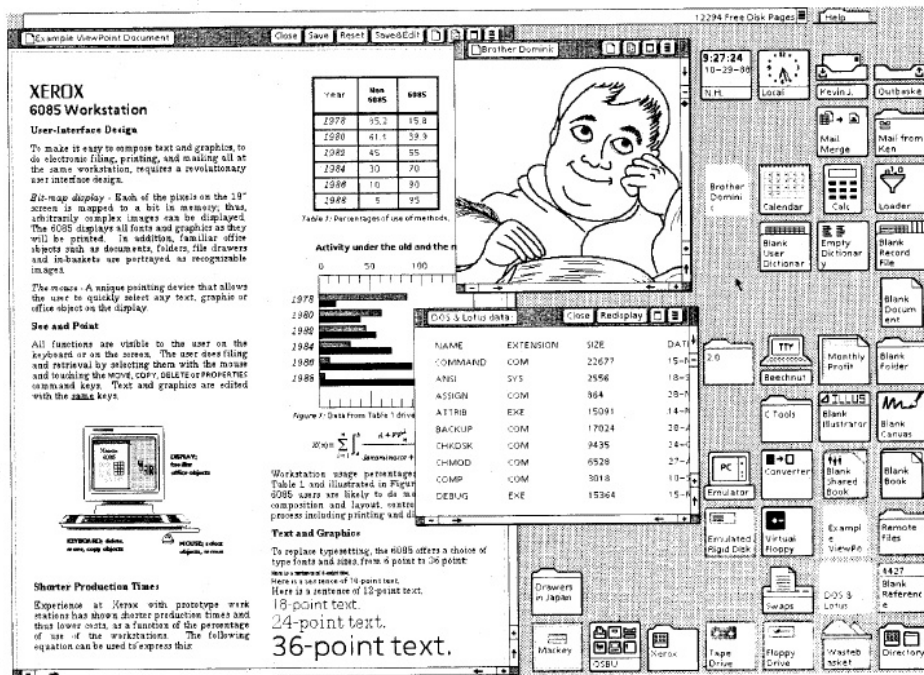


Abbildung 1.2: Der Xerox Star 8010 Desktop von 1981. Auf der linken Seite ist ein Fenster mit dem WYSIWYG Text Editor geöffnet, rechts kann man die Datei und Verzeichnis-Symbole auf dem Desktop erkennen. (Quelle: <http://www.digibarn.com>)

Die Forscher von Xerox PARC gaben dem bis dahin vorwiegend textbasierten Computer ein neues, graphisches Gesicht. Der Benutzer konnte erstmals Datei-Operationen durchführen, indem er mit der Maus Icons auf einem virtuellen Desktop verschob und die Eigenschaften der Daten, die durch diese Icons repräsentiert wurden, veränderte. Generische Befehle wie Kopieren, Löschen und Einfügen wurden auf spezielle Funktionstasten gelegt und standen damit direkt zur Verfügung. Die Bearbeitung von Dokumenten in einem Vorschau-Modus (*What You See Is What You Get* — WYSIWYG) erlaubte erstmals den direkten Eingriff in das Layout (s. Abbildung 1.2).

Die graphische Oberfläche versteckte die technischen Details des Betriebssystems und bot dem Benutzer graphische Metaphern, über die er auf die relevanten Funktionen zugreifen konnte [CSIK⁺82, JRV⁺er]. Schon bald folgten mit Apple LISA und dem Macintosh (1983/84) sowie Microsoft Windows 1.0 (1985) weitere Hersteller mit Systemen mit sog. WIMP (*Windows, Icons, Mouse, and Pointer*) GUI. Sie brachten günstige Geräte auf den Markt, die von den Menschen schnell angenommen wurden und rasche Verbreitung fanden. Der Personal-Computer mit Bildschirm, Maus und Tastatur und WIMP GUI etablierte sich zu einem neuen Standard, der noch heute vorherrscht.

Der Personal-Computer stellt heute die meist verbreitete Plattform dar und hat dabei in seiner Konfiguration mit Bildschirm, Maus und Tastatur im Vergleich zum Xerox Star in erster Linie qualitative Verbesserungen bezüglich Leistung und Ergonomie erfahren, ist in der Benutzung aber nahezu gleich geblieben. Auch die GUI ist in ihren grundlegenden Eigenschaften seit dem Xerox Star weitgehend unverändert geblieben. Komponenten wie Menüs, Zugriff auf Funktionen und Datei-Operationen unterscheiden sich zwischen den Herstellern hauptsächlich im Design und haben sich v. a. in der graphischen Qualität weiterentwickelt (s. Abbildung 1.3). Die Bedienkonzepte der Anfangsjahre finden sich aber noch größtenteils wieder [Enc03].

Aufgrund der Tastatur und Maus ist der Benutzer bei der Arbeit mit dem Computer meist auf eine Unterlage angewiesen. In der Regel geschieht die Arbeit deshalb meist im Sitzen an einem Schreibtisch. Der Nutzungskontext eines Personal-Computers ist damit recht klar eingeschränkt und wenig variabel (s. Tabelle 1.1).

Laut Statistischem Bundesamt besaßen im Jahr 2002 ca. 55% aller Haushalte und 71% aller Unternehmen in Deutschland mindestens einen solchen Personal-Computer. Das bedeutet, dass über

die Hälfte der Menschen in Deutschland mit der Benutzung graphischer Benutzungsoberflächen und WIMP Systemen vertraut sind.

Eigenschaft	Beschreibung
Nutzungskontext	Arbeit oder Spiel, sitzend an einem Schreibtisch
Ausgabegerät	Bildschirm (VGA und größer)
Eingabegerät	Tastatur, Maus
Interaktionsmetapher	Graphische WIMP Systeme

Tabelle 1.1: Charakteristika des Personal-Computers

Die Vorteile dieser Homogenität in Ausstattung und Nutzung sind offensichtlich und haben zu einem großen Teil zur enormen Verbreitung von Computern in den letzten zwanzig Jahren beigetragen.

Für den Benutzer eines Personal-Computers bedeuten die einheitlichen Eingabegeräte (Maus und Tastatur) und Oberflächenkonzepte (WIMP) eine Reduzierung des Einarbeitungsaufwandes, sowie eine geringere kognitive Belastung während der Nutzung. Sowohl der Wechsel zwischen Computern als auch der Wechsel zwischen Programmen ist durch die graphische Metapher erleichtert.

Das Wissen über die Benutzung von Maus und Tastatur ist auf heutigen Rechnern fast universell wieder verwendbar. Auch die Elemente und Konzepte der Bedienoberfläche (Fenster, Menüs, Schaltflächen, etc.) können in ähnlicher Form stets wiederentdeckt werden. Bietet eine Anwendung ausreichend Hilfen und Hinweise an, kann deren Funktionalität im GUI meist vom Benutzer explorativ erschlossen werden. Die kognitive Belastung während der Benutzung wird v. a. dadurch reduziert, dass Funktionen in Form von Menüeinträgen oder Schaltflächen in der Benutzungsoberfläche vorhanden sind. Dort können sie über die grundlegende Aktionen wie Mausklick oder Tastendruck (sog. Eingabeprimittiven) aufgerufen werden. Das Memorieren von Befehlen ist also nicht notwendig (zu einer Diskussion der Interaktionsmetaphern, siehe [Shn98]).

Die Einführung und Verbreitung von WIMP Systemen hat also für den Benutzer insgesamt zu einer Verbesserung der Bedienbarkeit und einer Reduzierung des Lernaufwandes geführt.

Den Entwicklern von Anwendungen für den Personal Computer bietet das einheitliche Interaktionskonzept, sowie der klar umrissene Nutzungskontext ebenfalls große Vorteile. Die Konzeption für die Benutzungsoberfläche und die Benutzerführung kann einmal für alle adressierten Plattformen geschehen, da sich verfügbarer Bildschirmplatz und Eingabemittel meist gleichen.

Die Implementierung von WIMP Oberflächen wird in modernen objektorientierten Sprachen meist durch vordefinierte Bibliotheken erleichtert, die meist aus ähnlichen Komponenten zusammengesetzt sind (z.B. *Microsoft Foundation Classes*, *Apple Application Kit* oder *Sun Microsystems Java Swing*). Graphische Designtools unterstützen die graphisch-interaktive Erzeugung von Benutzungsoberflächen.

Mehrfacher Entwicklungsaufwand für verschiedene Plattformen wird dadurch reduziert und der Schwerpunkt der Oberflächenentwicklung auf die Konzeption anstatt auf die Implementierung gelegt.

Dadurch, dass sich ein großer Teil der Forschung auf dem Gebiet der Mensch-Maschine-Interaktion auf die vorherrschenden WIMP Systeme konzentrieren konnte, liegen heute umfassende Erkenntnisse zur Verbesserung und Anpassung von graphischen Benutzungsoberflächen vor. Unter den Begriffen *User Interface Design* und *Usability Engineering* bieten zahlreiche Forschungsarbeiten Erkenntnisse und Lösungsansätze zu psychologischen, ergonomischen und soziologischen Implikationen der Mensch-Maschine-Interaktion mit WIMP Systemen und deren Erfassung [GC87, Wan93, Shn98]. Hersteller geben Hilfen und Richtlinien zur optimalen Gestaltung von Oberflächen vor und schreiben so teilweise herstellerübergreifende gestalterische Standards fest [App03, Mic03]. Gesetzgeber und Standardisierungsgremien haben durch Vorschriften und Richtlinien allgemeingültige Standards zur optimierten Gestaltung graphischer Oberflächen geschaffen [ISO01, Uni03].

WIMP Systeme haben auf diese Weise einen sehr hohen Grad der Reife und Erprobtheit erreicht,



Abbildung 1.3: Der Microsoft Windows XP Desktop von 2001. Mehrere Fenster sind übereinander geöffnet. Am unteren Rand die so genannte Task-Leiste mit der Start-Schaltfläche über die auf alle Funktionen zugegriffen werden kann. (Quelle: <http://www.aci.com.pl/mwichary/guidebook>)

welcher es ermöglicht, im Spannungsfeld zwischen Innovationsdruck und technischen Anforderungen dennoch einen hohen Grad an Benutzbarkeit und Akzeptanz zu erreichen.

Wirtschaftlich wirken sich oben genannte Faktoren, wie einfache Benutzbarkeit, Reduzierung des Entwicklungsaufwandes und Standards sehr positiv aus. Der Schulungsaufwand sinkt dadurch für Unternehmen, die Informationstechnologie einsetzen, stark. Fehlerbedingte Verluste gehen durch die bessere Benutzbarkeit zurück. Softwarehersteller profitieren unmittelbar durch die Einsparung bei der Entwicklung und die Vereinfachung der Pflege.

Das Konzept des Personal Computers, die Homogenität des — meist professionellen — Nutzungskontextes und die intuitive, konsistente und universell einsetzbare Interaktionsmetapher der WIMP Systeme haben zu der erfolgreichen Verbreitung der Informationstechnologie beigetragen.

In seinem wegweisenden Buch von 1989 hebt Jakob Nielsen die Chancen eines homogenen und in sich schlüssigen Oberflächenkonzeptes hervor [Nie89]. Nielsen führt hierzu den Begriff *Consistency* (engl. Konsistenz) ein, mit welchem er die einheitliche und vorhersagbare Verwendung von Oberflächeninhalten und Funktionen innerhalb einer Anwendung und über mehrere Anwendungen oder Plattformen hinweg bezeichnet. Nielsen stellt klar, dass der Begriff der Konsistenz ein sehr vielschichtiger Begriff ist und viele Dimensionen und Ebenen umfasst, nicht zuletzt deshalb bleibt er in seinem Buch eine klare und präzise Definition schuldig. Nielsen zeigt, dass Konsistenz, zumindest im Kontext der Gestaltung der Mensch-Maschine-Schnittstelle, selbst kein konsistentes Konzept ist und, dass der offensichtliche Vorteil einer einheitlichen und vorhersagbaren Gestaltung der Interaktion ebenfalls Gefahren in sich birgt [Nie89, S. 4f.]. Grudin [Gru89] stellt in ihrer Replik auf Wiecha et al. [WBBG89] klar, dass eine strenge Einhaltung von Konsistenz zu einer Behinderung des Benutzers führen kann, wenn eine inkonsistente aber günstigere Lösung ausgeschlossen wird. In der vorliegenden Arbeit wird der Begriff der Konsistenz vor allem im Kontext der plattformübergreifenden Entwicklung von Benutzungsoberflächen gebraucht werden und implizit als Ziel der plattformübergreifenden Entwicklung verwendet werden.

1.1.2 Jenseits des Personal Computers

Das Vordringen der Informationstechnik in immer weitere Teile unseres Lebens hat allerdings auch dazu geführt, dass diese den klassischen Nutzungskontext des Personal-Computers immer weiter verlassen hat und neue Anwendungsfelder eröffnete, wie die mobile Nutzung und den Einsatz von Computern im Haushalt, Unterhaltungselektronik, Automobil-Bordsystemen, öffentlichen Räumen, und vielen anderen Orten. Ben Shneiderman betrachtet diese Entwicklung als ein soziales Phänomen:

“In the old computing, computer usage was usually defined as a solitary experience, a concept that was encouraged by the term *personal computer*. But turning outward to focus on relationships led me to a fresh place where *family computer*, *corporate community*, or *civic network* might be appropriate terms.” [Shn03]

Da der klassische Personal Computer und seine WIMP Benutzungsoberflächen u.a. wegen seiner Größe, den wenig flexiblen Eingabegeräten, seinem Ressourcenverbrauch für viele dieser Anwendungsfelder ungeeignet ist, wurden neue für die Anwendungsfelder optimierte Geräte entwickelt. Mit ihnen entstanden neue Interaktionsmetaphern. Die klassischen Fenster-basierten Benutzungsoberflächen mussten z.T. angepasst werden oder völlig neuen Metaphern weichen.

Die Ära des Personal Computers neigt sich dem Ende entgegen. Viele Vorteile der homogenen Landschaft der Informationstechnik scheinen damit verloren zu gehen, mit allen Folgen und Chancen die dies mit sich bringt.

Gegen Ende der Neunziger Jahre trat eine tiefgreifende Veränderung in der Informationstechnik ein, deren Folgen bis heute noch nicht abzuschätzen sind: die Mobilisierung des Computers. Zurückzuführen ist diese Entwicklung vorwiegend auf drei Faktoren:

- **Das Internet:** Die Entwicklung des Internets zu einem Medium, welches auch immer stärker privat genutzt wird, hat die Bedeutung des Computers weg von dem reinen Arbeitsgerät hin zu einem Kommunikations- und Recherchewerkzeug verändert. Laut Statistischem Bundesamt verfügten im Jahre 2004 95% der Unternehmen ab 10 Beschäftigten über einen Internetzugang, und 50% der 16- bis 74-jährigen Bevölkerung gingen regelmäßig, d. h. mindestens einmal pro Woche, online [Sta05].
- **Miniaturisierung:** Die fortschreitende Miniaturisierung der Prozessor-, Display- und Speichertechnik ermöglicht die Entwicklung immer kleinerer Endgeräte mit immer längeren Bereitschaftszeiten. Mobilcomputer, die in Leistung und Funktionsumfang den älteren Personal-Computern kaum nachstehen (z.B. der Intel XScale Prozessor für Palmtops arbeitet mit 400 MHz Taktung) sind heute preiswert verfügbar.
- **Drahtlose Datenübertragungstechniken:** Die Verfügbarkeit leistungsfähiger drahtloser Datenübertragungstechniken, wie den Mobilfunk-Netzen (GSM, GPRS, seit 2004 auch UMTS) oder lokalen Funknetzen (IEEE 802.11x oder Bluetooth®) macht die mobile Nutzung von neuen Kommunikations- und Informationswegen wie dem Internet (s.o.) möglich.

Die Voraussetzungen für die verbreitete Verwendung mobiler Informationsdienste sind geschaffen: mit dem Internet steht heute ein offenes und leistungsfähiges Medium zur Verfügung und mit den Geräten und Datennetzen die notwendige Infrastruktur. Es ist lediglich eine Frage der Zeit bis die Anwendungen zur Verfügung stehen, um die mobile Nutzung von Informationstechnik über die Nutzung von Sprachdiensten hinaus in breitem Maße zu etablieren. Das Thema der mobilen kontextbezogenen Dienste gehört zu den zentralen Forschungsthemen in der Informationstechnik des frühen 21. Jahrhunderts [IST03, IST02, Enc01, Wah04]. Die Vision des allgegenwärtigen „ubiquitous computing“ [Wei91] führt bis hin zur Auflösung des Computer Interfaces in der ‚smarten‘ Umgebung und lässt ahnen dass die Zukunft des User Interfaces eine weit größere Vielgestaltigkeit bieten wird, als heute überhaupt vorstellbar ist (s. Abbildung 1.4) [RH04a].

Selbstverständlich waren während der vergangenen zwanzig Jahre ständig technische Innovationen entwickelt worden, die über den Personal Computer hinausgingen, wie z.B. erste tragbare Minicomputer wie der Newton von Apple oder der Fujitsu Stylus, oder alternative Eingabemethoden wie die Sprachsteuerung oder immersive Metaphern. Keine dieser Lösungen führte jedoch zu einem Durchbruch oder dauerhaftem Erfolg. Entweder die vorhandenen Konzepte der WIMP Systeme wurden



Abbildung 1.4: Die Vision der Ambient Intelligence, die den Menschen unterstützt und ihm kontext-bezogen Informationen und Dienste anbietet.

übernommen (s. Fujitsu Stylus) oder sie blieben Spezialanwendungen (VR-Systeme, Sprachsteuerung) oder starben gar ganz aus (Apple Newton). Erst mit der „mobilen Revolution“ im ausgehenden Zwanzigsten Jahrhundert waren alle Voraussetzungen für eine weitgreifende Veränderung gegeben. Mit ca. 400 Millionen verkauften Mobiltelefonen im Jahre 2002 (Quelle: Gartner) ist hier ein wichtiger Markt für Gerätehersteller sowie Infrastruktur-, Dienste- und Software-Anbieter entstanden.

Für die Interaktion mit dem Computer hat dessen Mobilisierung weitreichende Folgen: der Nutzungskontext kann nun stark variieren. Anstatt am Schreibtisch sitzend, wie beim Personal-Computer, interagiert der mobile Nutzer in allen erdenklichen Situationen: sitzend, stehend, laufend und fahrend. Je nach Anforderung kann bei Mobilgeräten das Display in Auflösung, Größe und Farbe stark variieren. Auch die Eingabemittel müssen den Gegebenheiten (Einhand-Bedienung, keine Unterlage, etc.) angepasst werden, d. h. der Standard Maus und Tastatur muss neuen Eingabegeräten, wie Stift, Zahlenfeld, Jogdial aber auch anderen Modalitäten wie z.B. Laut- bzw. Spracheingabe weichen. Die Benutzungsoberfläche ist aufgrund der Einschränkungen bei Ein- und Ausgabe selten an die bekannten WIMP Systeme angelehnt, und vielmehr als menübasiertes System ausgelegt. Zwischen den Herstellern können sich die meist proprietären Benutzungsoberflächen stark unterscheiden. Die Homogenität der Interaktions-Charakteristika des Personal-Computers und deren Vorteile, die oben (s. 1.1.1) besprochen wurden, ist in der mobilen Informationstechnik nicht mehr vorhanden (s. Tabelle 1.2), die erarbeitete Konsistenz in Benutzerführung, Oberflächengestaltung und Interaktionsmetaphern ist durch das Hinzukommen neuer Plattformen in Frage gestellt.

Eigenschaft	Beschreibung
Nutzungskontext	alle möglichen Situationen
Ausgabegerät	Bildschirm (SW/Farbe, Auflösung bis VGA und größer), Sprache
Eingabegerät	Stift, Zahlenfeld, Tastatur, Sprache, ...
Interaktionsmetapher	WIMP angepasst, Menü, andere ...

Tabelle 1.2: Bandbreite der Interaktions-Charakteristika von Mobilgeräten.

So vorteilhaft die Homogenität der Personal-Computer Plattform sich auf die Entwicklung und Verbreitung der Informationstechnik ausgewirkt hat, so negativ können die Folgen der Diversifizierung durch die Vielfalt mobiler Endgeräte und Plattformen sein. Ein Benutzer muss sich, aufgrund der grundlegenden Unterschiede in Oberflächen und Eingabemetaphern beim Wechsel von Geräten umstellen. Es bedarf teils erheblichen Aufwandes um die Bedienung eines neuen Gerätes zu erlernen (z.B. Texteingabe mit Graffiti auf PalmOne-Geräten). Hat der Benutzer bereits Erfahrungen mit einer Anwendung (z.B. einem Kalender) auf einem Gerät, bedeutet dies nicht, dass er eine gleichwertige Anwendung auf einem anderen Gerät problemlos bedienen kann. Eine explorative Voruntersuchung im Rahmen dieser Arbeit zeigte die Probleme, welche Benutzer beim Wechsel zwischen der Anwendung Microsoft Outlook für den Personal-Computer und Microsoft Pocket Outlook für den Pocket PC hatten (die Ergebnisse dieser Untersuchung werden im Detail an anderer Stelle in dieser Arbeit vorgestellt werden, s. 2.2). Die Möglichkeit des Transfers von Vorwissen zwischen Geräten und Plattformen scheint sehr eingeschränkt. Vielmehr scheinen widersprüchliche Interaktionskonzepte den Wechsel zusätzlich noch zu erschweren.

Ein Entwickler, der Anwendungen für verschiedene Geräte konzipieren muss — wir sprechen von plattformübergreifender oder *multi-plattform* Entwicklung — braucht spezifische Kenntnisse für jede der Plattformen die adressiert werden soll. Aufgrund der Unterschiede in Display-Größe, Farbtiefe und Auflösung, sowie Interaktionsmittel, müssen die Benutzungsoberflächen (aufgrund der mangelnden Kompatibilität der Betriebssysteme in der Regel die gesamten Anwendungen) meist speziell für die Geräte entwickelt werden. Wiederverwendung und Synergieeffekte treten bei der Entwicklung für mehrere Plattformen nur selten auf.

Usability Regeln und Konzepte lassen sich nur auf sehr allgemeinem Niveau über verschiedene Plattformen definieren. Die Gestaltungsgrundsätze für bedienbare Benutzungsoberflächen wie z.B.

die der DIN EN ISO 9241 [ISO01], *Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Erwartungskonformität, Steuerbarkeit, Fehlertoleranz, Individualisierbarkeit und Lernförderlichkeit* sind natürlich anwendbar, jedoch können die Erfahrungen aus der Gestaltung der WIMP Oberflächen des Personal-Computers nur bedingt übertragen werden. Die Untersuchung der Bedienbarkeit von solchen Mobilgeräten ist deshalb noch weitgehend Neuland [Wei02].

Wirtschaftlich stellt die parallele Entwicklung und Pflege von Anwendungen für verschiedene Plattformen eine erhebliche Mehrbelastung dar. Hatten plattformunabhängige Anwendungskonzepte wie Java von Sun Microsystems den redundanten Entwicklungsaufwand für den Personal Computer gerade reduziert, führen die heterogenen Oberflächen und Interaktionskonzepte der Mobilgeräte nun wieder dazu, dass erheblicher Aufwand in parallele Entwicklungen fließen muss. Unternehmen müssen Support und Schulung für jede eingesetzte Plattform anbieten. Das bedeutet, dass ein Außendienstmitarbeiter, der im Büro einen Personal Computer und unterwegs einen Handheld Computer nutzt, auf jedem der beiden Geräte gesondert geschult werden muss.

Dieser Aufwand, der mit der plattformübergreifenden Entwicklung und Nutzung verbunden ist, führt zu einer Zurückhaltung bei Einsatz und Entwicklung. Soll die Erfolgsgeschichte des Personal-Computers für die Mobile Nutzung fortgeschrieben werden, müssen Lösungen entwickelt werden, die diesen Aufwand reduzieren.

Aus diesem Grund ist ein Ziel dieser Arbeit die Erarbeitung eines Konzeptes zur Unterstützung der Entwicklung von plattformübergreifenden Benutzungsoberflächen. Die Aufrechterhaltung von vereinheitlichten, für den Nutzer transparenten und vorhersagbaren und einfacher Interaktionskonzepten ist ein wichtiges Ziel in der weiteren Entwicklung plattformübergreifender Systeme.

Hypothese 1

Die Entwicklung plattformübergreifender Systeme kann durch benutzerzentrierte und psychologisch motivierte Methoden vereinfacht und verbessert werden.

Das Vordringen der Informationstechnik in unseren Alltag, in Verkehr und Verwaltung, hat zur Folge, dass in vielen Bereichen des öffentlichen Lebens Dienste und Waren zunehmend über den elektronischen Weg vertrieben werden. Behörden stellen Formulare im Internet zur Verfügung. Verkehrsbetriebe verkaufen Fahrkarten nicht mehr an Schaltern oder beim Schaffner sondern an Automaten mit Touchscreen und elektronischer Bezahlung. Kaufhäuser ersetzen die Informationsschalter durch interaktive Informationsterminals. Banken bieten Online-Banking zu vergünstigten Konditionen und wickeln den Privatkunden-Verkehr über Bankautomaten, Überweisungsterminals und sogar Kredit-Terminals ab. Es wird immer wichtiger für den Bürger in unserer Informationsgesellschaft, derartige Dienste nutzen zu können. Der Vorteil durch die schnellere und bequemere Erledigung über das Internet erscheint noch Luxus, doch schon der Zugang zu Bargeld kann problematisch werden, hat man keine Möglichkeit, Bankautomaten zu nutzen (s. Abbildung 1.5).

In diesem Zusammenhang wird klar: Informationstechnik hat eine gesellschaftliche Verpflichtung. Die Zugänglichkeit von Informationstechnik, und damit auch die Entwicklung von Benutzungsoberflächen, die von jedem bedient werden können, muss ein zentraler Aspekt moderner Informationssysteme sein. *Accessibility* (engl. für Zugänglichkeit) wird heute meist im Zusammenhang mit behinderten Nutzern betrachtet. *Accessibility* wird meist als optionale Eigenschaft betrachtet, die auf die eigentlichen Anwendungen aufgesetzt werden. Selten wird *Accessibility* als integraler Teil der Informationstechnik selbst betrachtet.

Wie Stephanidis [Ste01] in seiner Forderung nach *Universal Accessibility* feststellte, ist die Reduzierung von *Accessibility* auf *Zugang für behinderte Nutzer* zu kurz blickend. Stephanidis warnt davor, dass die technische Entwicklung und das schrittweise Vordringen der Informationstechnologie in unser Leben dazu führt, dass immer mehr Menschen Probleme mit der Zugänglichkeit von solchen Systemen bekommen werden „[...] the range of the population which may gradually be confronted with accessibility problems extends beyond the population of disabled and elderly users [SAS98].“ *Accessibility* fordert von Systemen die Fähigkeit, sich an die Bedürfnisse und Möglichkeiten des Nutzers, an den Nutzungskontext und die Aufgabe und an die technische Plattform anzupassen.

Soll nun also ein informationstechnisches System die Fähigkeit haben, sich an die vom Nutzer verwendete technische Plattform anzupassen, bedeutet dies nichts anderes als z.B. die Möglichkeit zur



Abbildung 1.5: Probleme bei der Zugänglichkeit von Informationstechnik im öffentlichen Raum, Beispiel: Bankautomat. Die Nutzung von Informationstechnik ist für Menschen mit Behinderungen häufig erschwert oder gar unmöglich.

plattformübergreifenden Bereitstellung des UI. Fokussiert man hierbei den mobilen Nutzer, muss Informationstechnik auf Mobilgeräte abbildbar sein. Fokussiert man den Nutzer mit körperlichen oder geistigen Einschränkungen muss die Abbildung in verschiedene Modalitäten oder auf verschiedene Interaktionsmetaphern möglich sein um so die spezifischen Einschränkungen der Nutzer ausgleichen zu können. Oviatt sieht in multimodalen UI die Chance den individuellen Fähigkeiten der Nutzer optimal Rechnung zu tragen:

“[...] multi modal interfaces have the potential to accommodate a broader range of users than traditional interfaces — including users of different ages, skill levels, native language status, cognitive styles, sensory impairments, and other temporary illnesses or permanent handicaps.” [Ovi03]

Ein System, welches die Entwicklung von UI auf verschiedene Endgeräte und Modalitäten ermöglicht, entspricht der Forderung nach Universal Accessibility. Dies bedeutet, es betrachtet die Zugänglichkeit als elementaren Bestandteil der Informationstechnik und stellt sich damit der gesellschaftlichen Verantwortung dieser Technik.

Hypothese 2

Plattformübergreifende Benutzerschnittstellen berücksichtigen die Notwendigkeit der Zugänglichkeit von Informationstechnik für alle. Ein Konzept plattformunabhängiger Benutzerschnittstellen dient somit Menschen mit und ohne Behinderungen bei der selbstständigen Nutzung von Informationstechnik.

1.1.3 Zusammenfassung

Wie gezeigt wurde, befindet sich die Entwicklung weg von der bislang vorherrschenden Personal Computern und ihren WIMP basierten Benutzerschnittstellen, hin zu einer vielgestaltigen Koexis-

tenz verschiedener Endgeräte, Anwendungskontexte und Interaktionsmetaphern bereits in vollem Gange. Hauptverantwortlich hierfür sind vor allem zwei Tatsachen: (a) der Anteil der mobilen Nutzung von Informationstechnik und der dafür verfügbaren spezialisierten Endgeräte steigt, und (b) die Anforderungen an die Benutzbarkeit dieser Systeme steigt in dem Maße ihrer Verbreitung. Die Geräte müssen sich an den Benutzungskontext und die Fähigkeiten des Benutzers anpassen, sei es, dass er sich im Auto befindet und es deswegen vorzieht, über Sprache zu interagieren, sei es, dass er aufgrund mangelnder Sehfähigkeit (z.B. aufgrund einer Behinderung) auf alternative Modalitäten angewiesen ist.

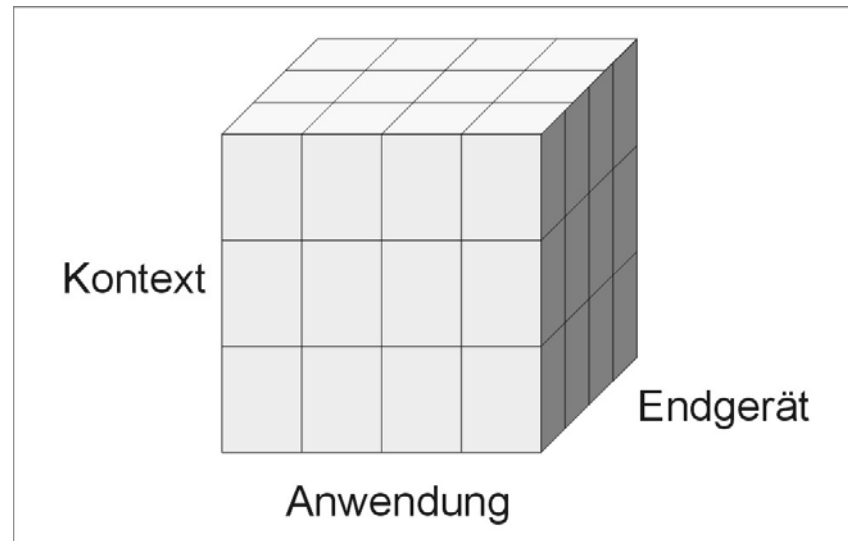


Abbildung 1.6: Anwendungsfälle kontext- und geräteübergreifender Anwendungsentwicklung.

Wie Abbildung 1.6 zeigt, werden Applikationen also in zunehmendem Maße in verschiedenen Kontexten, in verschiedenen Anwendungsfällen und auf verschiedenen Endgeräten eingesetzt werden, d. h. Anwendungen müssen in der Lage sein, sich hierauf ohne großen Aufwand anpassen zu lassen. Anwender werden in zunehmendem Maße mit identischen Anwendungen auf verschiedenen Endgeräten und in verschiedenen Kontexten interagieren, d. h. der Umstieg zwischen den Plattformen sollte sich möglichst einfach und intuitiv gestalten. Es werden Werkzeuge benötigt, welche die Entwicklung und Pflege solcher plattformübergreifender Anwendungen unterstützen.

1.2 Lösungsansatz

In der vorliegenden Arbeit soll—ausgehend von der im letzten Abschnitt diskutierten Problematik der wachsenden Variabilität der informationstechnischen Endgeräte—Methoden entwickelt werden, mit Hilfe derer die Entwicklung von Anwendungen für beliebige Endgeräte und Modalitäten verbessert, vereinfacht und beschleunigt werden kann. In Anlehnung an die Arbeit von Seffah und Javahery [SKD01, SJ04a] werden diese plattformübergreifenden User Interfaces im Rahmen dieser Arbeit als *Multiple User Interfaces* bezeichnet werden. Die Definition von *Multiple User Interfaces* geht um einiges über das hinaus, was gemeinhin unter plattformübergreifend verstanden wird und trifft somit das Anliegen dieser Arbeit besser. Seffah und Javahery [SJ04a] definieren *Multiple User Interfaces* als interaktives System welches:

- Zugang zu Informationen und Diensten mittels verschiedener Computer Plattformen ermöglicht;
- verschiedene Sichten derselben Information auf den verschiedenen Plattformen bietet;
- Dienste für einzelne Benutzer oder Gruppen von Benutzern koordiniert.

Um einen möglichen Lösungsansatz für die oben geschilderte Problematik aufzuzeigen, soll ein Framework konzipiert werden, welches den Entwickler solcher Multiple User Interfaces bei der

Arbeit unterstützen soll, indem es die Komplexität des Entwicklungsprozesses reduziert und die Beurteilung des Ergebnisses vereinfacht. Auf diese Weise soll der Aufwand beim Einsatz solcher Lösungen reduziert werden und ein deutlicher Beitrag zur möglichen Verbreitung multipler User Interfaces gemacht werden. Die Qualität der so erzeugten Oberflächen soll durch die Unterstützung innerhalb des Entwicklungsprozesses verbessert werden und die Vorteile dieses Ansatzes so mehr Menschen zugänglich gemacht werden.

Hieraus lassen sich die folgenden drei Kernziele für diese Arbeit ableiten:

Sicherstellung der Benutzbarkeit Der Entwickler erhält Unterstützung bei der Gestaltung benutzbarer plattformübergreifender Benutzerschnittstellen. Die Entwicklungswerkzeuge sollen den Gestaltungsprozess leiten und existierendes Wissen über Voraussetzungen benutzbarer Systeme, wie z. B. plattformübergreifende Konsistenz, in den Prozess einfließen lassen. Dies soll in Form von automatisierten Methoden erfolgen.

Reduzierung des Anpassungsaufwandes Es sind Methoden notwendig Anwendungsoberflächen einfach und schnell an neue Geräte anzupassen. Diese Methoden sollten möglichst transparent für den Entwickler automatisch oder unterstützt innerhalb des Darstellungssystems angewandt werden. Die Möglichkeit der Wiederverwendung existierender Lösung in Form von Design Patterns stellt eine weitere Lösung dar.

Reduzierung des Einarbeitungsaufwandes Die Entwickler von plattformübergreifenden Anwendungen sollen—soweit dies möglich ist—dieselben Entwicklungsmethoden verwenden können wie bislang. Dies bedeutet das der Einsatz von graphischen Entwicklungswerkzeugen oder erfahrungskonformen Anwendungsprogrammierungsschnittstellen (APIs) unterstützt werden sollen.

1.3 Abgrenzung und Einordnung der Arbeit

Ziel dieser Arbeit ist es Methoden zur Unterstützung bei der Entwicklung plattformübergreifender Benutzerschnittstellen zu erarbeiten. Im Vordergrund steht hierbei der Wunsch, die Prozesse zu verstehen, welche bei der Benutzung derselben Anwendungen auf verschiedenen Endgeräten ablaufen und wie die Benutzbarkeit solcher Systeme zu verbessern ist. Hieraus sollen Methoden abgeleitet und empirisch validiert werden, die zu einer Verbesserung der plattformübergreifenden Entwicklung beitragen.

Die Herangehensweise dieser Arbeit ist *benutzerorientiert*, wobei von den Gruppen der Entwickler, der Designer und der Endbenutzer des Systems ausgegangen wird. Dies bedeutet dass dieser Entwicklungsprozess zum einen als Prozess selbst, zum anderen in seinem Endprodukt, dem interaktiven System auf die Anforderungen seiner Benutzer optimiert wird.

Die *technische* Ableitung von Spezifikationsmethoden oder die Anwendung von *Modellen* zur Beschreibung von Interaktionen, Aufgaben und Benutzerrollen spielen bei der Anpassung von Benutzerschnittstellen auf den Nutzungskontext eine wichtige Rolle. Allerdings existieren hierzu bereits zahlreiche Arbeiten und Entwicklungsansätze, welche in Abschnitt 4 ausführlich diskutiert werden, so dass im Rahmen dieser Arbeit auf eine gesonderte Betrachtung verzichtet wurde. Wo möglich wurde auf die Kompatibilität des Systems mit modellbasierten Ansätzen und kontextadaptiven Methoden geachtet. Der Fokus der Arbeit lag jedoch auf der Anpassung an verschiedene Endgeräte, so das andere Kontext-Parameter hier nur teilweise (z. B. in der Validierung des Darstellungssystems) berücksichtigt wurden.

In der vorliegenden Arbeit werden zum einen Methoden der Entwicklungsunterstützung zum anderen Methoden der Bewertung von Benutzerschnittstellen auf ihre Benutzbarkeit entwickelt. Im Vordergrund steht hierbei das Kriterium der *Konsistenz*, welches allgemein als zentrales Kriterium für den erfolgreichen Transfer von Wissen angesehen wird (s. Abschnitt 3). Andere Kriterien der Benutzbarkeit spielen in der Betrachtung eine Rolle, werden jedoch nur nachrangig behandelt.

Der Fokus dieser Arbeit liegt klar auf der Gestaltung graphischer Benutzerschnittstellen und *graphisch-interaktiver* Methoden diese zu unterstützen. Der Aspekt der multimodalen Interaktion wird im Kontext der universellen Zugänglichkeit diskutiert und im Rahmen eines prototypischen Systems auch umgesetzt, stellt jedoch keinen zentralen Aspekt dieser Arbeit dar.

Das in dieser Arbeit entwickelte und prototypisch implementierte Darstellungssystem für plattformübergreifende Benutzerschnittstellen verwendet in seinem Abstraktionskern einen existierenden Standard für die geräte- und darstellungunabhängige *Beschreibung von Benutzerschnittstellen*. Aus technischen Gründen musste dieser in eine Host-Sprache eingebettet werden, welche im Rahmen dieser Arbeit entwickelt wurde. Diese Arbeit hat nicht zum Ziel eine neue Beschreibungssprache für User Interfaces [NM03, NM04, AP99] oder Konzepte zur deren abstrakter Repräsentation, Migration [BB03] oder Aufteilung auf mehrere Endgeräte zu entwickeln, sondern versucht bei der technischen Umsetzung weitestgehend existierende standardkonforme Lösungen zu verwenden und gemäß der Entwicklungsmethodik zu erweitern.

1.4 Gliederung

Die vorliegende Arbeit beschreibt die Konzeption, Entwicklung und Validierung von Methoden der Entwicklungsunterstützung bei der Gestaltung plattformübergreifender Benutzerschnittstellen. Die Arbeit gliedert sich in acht Kapitel, deren Inhalt hier kurz zusammengefasst werden soll.

Die *Einführung*, die mit diesem Überblick schließt, führt zunächst in die Problematik der Entwicklung von Anwendungen für verschiedene Endgeräte ein und motiviert die Potentiale der universellen Benutzbarkeit und zukünftiger Computer-Metaphern wie die der Ambient Intelligence sowie die Notwendigkeit der Unterstützung bei der Entwicklung derartiger Systeme. Ein kurzer Abriss des Lösungsansatzes, der mit dieser Arbeit verfolgt wird sowie eine Abgrenzung und Einordnung dieser Arbeit schließen sich an.

Der zweite Teil dieser Arbeit enthält eine umfassende *Analyse der Aufgabenstellung*. Diese besteht zunächst aus einer empirischen Studie zur Exploration der Probleme, die für einen Benutzer plattformübergreifender Systeme entstehen. Der Transfer von Wissen zwischen zwei Versionen einer kommerziellen Anwendung für Desktop und Pocket PC wird anhand der auftretenden Fehler analysiert und klassifiziert. Anschließend werden anhand der Erfahrungen aus dieser Studie sowie anhand der Literatur Anforderungen an die Benutzbarkeit, an die Entwicklungsprozesse und an die technische Konzeption plattformübergreifender Entwicklung aufgestellt.

Die *Grundlagen*, auf welche die Konzepte dieser Arbeit aufbauen, werden im dritten Teil dieser Arbeit zusammenfassend beschrieben. Die kognitionspsychologischen Erkenntnisse zum Transfer von Wissen, zur Repräsentation und Aneignung von Wissen durch mentale Modelle sowie die Bedeutung der Konsistenz als Aspekt der Benutzbarkeit interaktiver Systeme werden hier ausführlich beleuchtet.

Anschließend werden die existierenden *Methoden der Entwicklung von Benutzerschnittstellen* zusammengefasst und Bezug auf die Anforderungen diskutiert. Generell werden hierbei Window-Systeme, User Interface Toolkits und User Interface Management Systeme (UIMS) unterschieden. Innerhalb der UIMS wird hier insbesondere auf die Bedeutung der modellbasierten Ansätze hingewiesen. Weiterhin werden existierende Ansätze der algorithmischen Bewertung von Eigenschaften graphischer Oberflächen durch verschiedene Metriken diskutiert.

Ein *Konzept plattformübergreifender Entwicklung* wird im fünften Teil dieser Arbeit entwickelt. In diesem zentralen Kapitel werden aufbauend auf den Grundlagen und den Anforderungen ein benutzerzentrierte Entwicklungsmethodiken in ein Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen überführt. Ausgehend von den Erkenntnissen der initialen Benutzerstudie und den kognitionspsychologischen Grundlagen des Wissenstransfers werden dann verschiedenen Strategien zur Abbildung von Darstellungen zwischen Endgeräten beschrieben und klassifiziert. Eine Methode zur Berechnung der transformationalen Konsistenz wird beschrieben.

Die *Realisierung der Entwicklungskonzepte* schließt sich an und gliedert sich in zwei Teile, die Realisierung des Darstellungssystems und die Einbindung des transformationalen Konsistenzmaßes in den graphische Gestaltungsprozess von User Interfaces.

Die *Systemvalidierung* umfasst drei Teile. Zunächst wird die empirische Validierung der Benutzbarkeit des Darstellungssystems beschrieben. Die Validierung der Transformationshypothese wird

anhand einer Replizierung des Effekts der Mentalen Rotation sowie einer Anwendung von Transformationsstrategien zur Anpassung einer kommerziellen Anwendung beschrieben. Zuletzt wird das transformationale Konsistenzmaß auf seine Validität bezüglich der Benutzbarkeit plattformübergreifender Anwendungen in einer Benutzerstudie überprüft.

Die Arbeit schließt mit einer *Zusammenfassung und Ausblick*, in welchen die wesentlichen Ergebnisse dieser Arbeit nochmals zusammenfassend hervorgehoben und neue Forschungsfragen formuliert werden.

2 Analyse der Aufgabenstellung und Ableitung von Anforderungen

Wir haben damals die Windows-Idee eigentlich für Kinder entwickelt. Raten Sie mal, wer heute alles mit Windows arbeitet?

— Alan Kay, Erfinder der Windows-Metapher, in einem Interview mit der ZEIT, 2002

2.1 Analyse der Aufgabenstellung

Ziel dieser Arbeit ist die Konzeption eines Entwicklungsmodells für plattformübergreifende oder auch multiple Anwendungsoberflächen (s. Abschnitt 1.2). Mithilfe dieses Modells soll der Aufwand bei der Entwicklung von Oberflächen für beliebige Endgeräte reduziert werden. Der Einsatz von multiplen Benutzerschnittstellen soll vereinfacht werden und die Qualität und Benutzbarkeit der so erzeugten Oberflächen soll verbessert werden um so die Vorteile und Chancen, wie sie in Abschnitt 1.1.2 geschildert werden, zu erreichen.

Bislang werden Anwendungsoberflächen speziell für eine Zielplattform, oder zumindest für eine bestimmte Untermenge von Plattformen entwickelt. Hierzu existieren bereits spezielle Werkzeuge, z. B. sog. *Interface Builder*, die den Entwickler durch die Möglichkeit der graphischen Bearbeitung unterstützen. Mithilfe von Leitfäden und vorgefertigten Elementen (sog. *Toolkits*) kann heute existierendes Design-Wissen ohne großen Aufwand in den Entwicklungsprozess integriert und genutzt werden. Andere Methoden, wie Systeme zur Generierung von Benutzerschnittstellen aus abstrakten Spezifikationen, haben sich in Anwendung bislang nicht durchsetzen können, obwohl diese Methoden insbesondere bezüglich der Einheitlichkeit der Oberflächen manuellem Design überlegen sind.

Tatsächlich ist die Entwicklung von Benutzungsoberflächen heute stark geprägt durch das Wechselspiel zwischen visuellem Design und der iterativen Validierung des Ergebnisses. Die Effizienz des Entwicklungsprozesses steigt mit dem Maße, in dem die der Aufwand zur Testung reduziert werden und die Qualität des anfängliche Designs verbessert werden kann.

Daraus folgt also, dass bei der Analyse der Anforderungen sowohl der *Entwicklungsprozess* selbst als auch der *Benutzungsprozess* betrachtet werden muss. Diese beiden Prozesse werden im Rahmen der folgenden Anforderungsanalyse getrennt betrachtet werden und zu dann einem gemeinsamen Anforderungskatalog zusammen mit den technischen Anforderungen integriert werden.

Ein wesentlicher Aspekt der plattformübergreifenden Entwicklung ist die anwendungsübergreifende Homogenisierung der Oberflächen. Abbildung 2.1 zeigt, wie diese beiden Aspekte miteinander in Bezug stehen. Neben der Möglichkeit dieselbe Anwendung auf verschiedenen Endgeräten nutzen zu können, berücksichtigt dieses Konzept auch die Möglichkeit, verschiedene Anwendungen derart zu vereinheitlichen, dass auch der Wechsel zwischen verschiedenen Anwendungen vereinfacht wird.

Denkt man zurück an den Fortschritt, den die WIMP-Metapher und die einheitlichen, meist betriebs-systemspezifischen Styleguides für die einheitliche Gestaltung von Oberflächen mit sich brachte, so ist dieser Vereinfachung auch bei den *Multiple User Interfaces* wünschenswert und möglich. Ein weiterer Aspekt hierbei ist die Personalisierung der Oberflächen, d. h. die personenspezifische Anpassung der Oberflächen über verschiedene Anwendungen hinweg, z. B. im Sinne der Zugänglichkeit durch die Vergrößerung von Schriften oder die Auslassung von Icons.

Aus diesen Überlegungen folgt also zum Zweiten, dass die Aufgabenstellung nicht nur die *plattformübergreifende* Benutzung und Entwicklung berücksichtigen muss, sondern auch die *anwendungsübergreifende* Vereinheitlichung und Adaptierbarkeit zentrale Punkte sind. Auch dieser zweite Faktor soll zur Ordnung der folgenden Anforderungsanalyse und im Anforderungskatalog berücksichtigt werden.

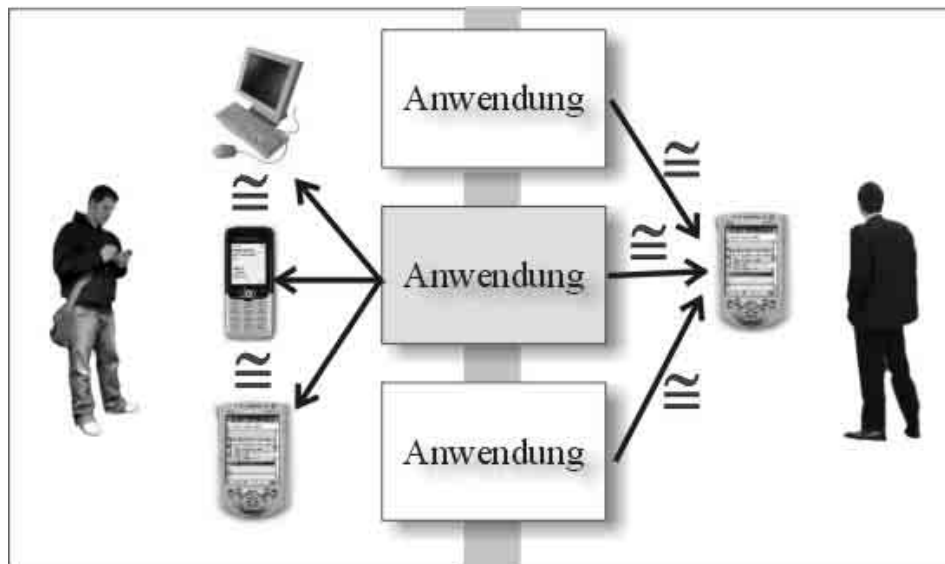


Abbildung 2.1: Plattformübergreifende vs. anwendungsübergreifende Entwicklung.

2.2 Empirische Untersuchung der Benutzeranforderungen

Um die Anforderungen der Benutzer an plattformübergreifende Anwendungen zu erfassen, wurde eine explorative Benutzerstudie durchgeführt. Die Teilnehmer dieser Studie sollten dieselbe Anwendung auf zwei verschiedenen Endgeräten benutzen. Die dabei gewonnenen Daten sollten genutzt werden, um die kognitiven Prozesse, welche bei dem Umstieg zwischen Geräten auftreten, zu analysieren und daraus resultierende Probleme zu identifizieren. Aus diesen Problemen sollten Anforderungen an die erfolgreiche Entwicklung von plattformübergreifende Anwendungen abgeleitet werden.

Ausgehend von der Annahme, dass Ähnlichkeiten der Anwendung auf beiden Plattformen den Transfer von Anwendungswissen unterstützen [SA89] und damit den Umstieg erleichtern, wurde in dieser Studie ein Vergleich auf verschiedenen Ebenen angestellt. Es wurde versucht, die Interaktion auf beiden Plattformen sowohl auf der Ebene der *Benutzerwahrnehmung* als auch auf der Ebene der *Aufgaben- und Anwendungsstruktur* zu analysieren, um Rückschlüsse auf den Transferprozess ziehen zu können.

2.2.1 Einleitung

Allgemein wird angenommen, dass der Benutzer einer Anwendung eine interne Repräsentation der Anwendung aufbaut, das so genannte mentale Modell. Das mentale Modell ist ein Konzept aus der kognitiven Psychologie [Cra43, JL83], welches erfolgreich auf die Domäne der Mensch-Maschine-Interaktion angewandt wurde [GS83] (s. Abschnitt 3.2). Dieses mentale Modell stellt nach Norman [Nor83] die mentale Repräsentation der Anwendung dar, welche aus dem Dialog mit der Anwendung entsteht. Sie dient dazu, Interaktionen zu planen und Hypothesen über die wahrscheinlichen Resultate aufzustellen.

Nach Norman [Nor83] liegen der Interaktion zwischen Mensch und Maschine neben dem mentalen Modell des Benutzers weitere Modelle zugrunde, deren Wechselwirkungen die Benutzbarkeit eines Systems in einem entscheidenden Maße bestimmen. Norman unterscheidet drei Arten von Modellen: das *mentale Modell* des Benutzers, das *konzeptuelle Modell* als genaue funktionale Beschreibung des Systems und das *System-Bild*, welches das System nach außen hin vermittelt. Ein gutes Design führt nach Norman dazu, dass das System-Bild so gestaltet ist, dass es das konzeptuelle Modell gut repräsentiert und dem Benutzer dabei hilft, ein adäquates mentales Modell aufzubauen. Dieses mentale Modell ist ein Resultat der Kommunikation zwischen dem Benutzer und der Maschine und entsteht im Laufe des Erlernens und der Benutzung.

Die Kommunikation zwischen dem Benutzer und dem System findet in zwei Richtungen statt. Zum einen muss der Benutzer dem System mitteilen, was es zu tun hat. Er muss hierzu die eigenen Ziele in die Aufgaben und Funktionen übersetzen, welche das System anbietet. Dann muss er diese Aufgaben mittels einzelner Aktionen und Kommandos auslösen, kontrollieren und steuern. Auf der anderen Seite muss der Benutzer in der Lage sein, die Ergebnisse des Prozesses auf dem System zu interpretieren und auf seine Aufgabe und Ziele zu übertragen und den Beitrag zur Erreichung der Ziele bewerten.

Verschiedene Modelle existieren, welche diese Mensch-Maschine Kommunikation abstrahieren und in verschiedene Ebenen unterteilen, wie z.B. das *Sprachmodell* von Foley und Van Dam [FD82], Normans *Gulf of Execution and Evaluation* [Nor88] und die *Ebenenmodelle* von Nielsen [Nie86] oder Moran [Mor81]. In Analogie zur natürlichen Sprache wird die Intention des Benutzers über verschiedene Ebenen transformiert und vermittelt. Nielsen [Nie86] unterscheidet die folgenden Ebenen der Abstraktion im Dialog zwischen Benutzer und System:

- *Aufgabenebene*: Definition der Aufgabe, welche der Benutzer erfüllen möchte.
- *Zielebene*: Definition der Ziele und Unter-Ziele, welche zur Erfüllung der Aufgabe erreicht werden müssen.
- *Semantische Ebene*: Definition der der Objekte, Attribute und Operationen, welche kommuniziert werden.
- *Syntaktische Ebene*: Definition des Vorgehens, der notwendigen Kommandos, etc.
- *Lexikalische Ebene*: Definition der konkreten Aktion, welche ein Kommando auslöst.
- *Physikalische Ebene*: Definition der physikalischen Realisierung der Aktion.

Zum Beispiel kann Mitarbeiter den Auftrag bekommen, einen Bericht zu kopieren (*Aufgabe*), das Ziel ist also n Exemplare des Berichtes zu erhalten (*Ziel*). Das bedeutet, dass er den Kopierer verwenden soll um den Bericht auf Papier zu vervielfältigen (*Semantik*). Hierzu benötigt er den Bericht und die Anzahl der Kopien, sowie die Information, ob die Kopie ein- oder zweiseitig erfolgen soll (*Syntax*). Die Kopierfunktion wird mit *Start* bezeichnet, die Duplexfunktion mit verschiedenen Seiten-Symbolen, usw. (*Lexikalisch*). Die Auslösung des Kopierfunktion erfolgt durch Drücken des Knopfes *Start* (*Physikalisch*).

In dieser Studie wurde versucht das mentale Modell des Benutzers von der Anwendung zu erfassen. Dann wurde untersucht wie sich dieses Modell zwischen den beiden Plattformen unterscheidet, um die Wahrnehmung der Anwendung auf beiden Plattformen vergleichen zu können. Weiterhin wurde versucht, die eigentliche Anwendungsstruktur, das konzeptuelle Modell zu analysieren und für beide Plattformen gegenüberzustellen. Die Kontrastierung sowohl der mentalen als auch der konzeptuellen Modelle sollte sich in der Art der Fehler [Rea90] widerspiegeln, welche die Anwender bei der Benutzer machten.

2.2.2 Erfassung des mentalen Modells

Das mentale Modell ist der Zugang des Benutzers zu dem System und aus diesem Grunde von zentralem Interesse bei der Untersuchung der kognitiven Prozesse beim Wechsel zwischen den Plattformen. Sollte es möglich sein, Zugang zu dieser Repräsentation erhalten und die Unterschiede dieser Repräsentation bei der Benutzung derselben Anwendung auf verschiedenen Plattformen zu erfassen, ließe sich daraus auf die Probleme beim Transfer des Anwendungswissens schließen. Jedoch, wie Van der Veer und Melguizo [VPM03] feststellen: „*The main problem is that, because mental models are in the mind of people, they can not be accessed directly*”.

Verschiedene Verfahren zur Erfassung des mentalen Modells, wie Interviews, *Thinking Aloud*, *Teach Back*, Protokolle und Beobachtungen, Beschreibung und Erklärung des Systems durch den Benutzer, Assoziationslisten und Beobachtung der Antizipation von des System-Verhaltens durch den Benutzer wurden angewendet um das mentale Modell des Benutzers zu erfassen [VPM03, Eng01]. Das interne mentale Modell muss externalisiert und dabei beschrieben werden. Im folgenden Abschnitt werden kurz die wichtigsten Methoden zur Erfassung des mentalen Modells beschrieben und diskutiert werden.

Die *Card Sorting* Methode wird verwendet um Elemente zu kategorisieren. Der Proband ordnet Begriffe, die auf einzelne Karten geschrieben werden, bzgl. ihrer semantischen Nähe oder anderen Kriterien (z.B. erinnerte Anordnung) an [ZAB06]. Die Elemente können hierbei bereits vorgegeben, oder im Rahmen einer Untersuchung gefunden worden sein. Die Methode wird insbesondere bei der Strukturierung komplexer Webseiten oder Menüs eingesetzt. Bei der Erfassung mentaler Modelle kann Card Sorting eingesetzt werden, um die semantische Nähe verschiedener Begriffe aufzuzeigen, die Anordnung kann dann mithilfe von Werkzeugen wie *Pathfinder*, *CardSword* oder *EZSort* (s. u.) analysiert werden.

Bei der *Thinking Aloud* Methode [NS72] wird der Benutzer aufgefordert, während der Durchführung einer Aufgabe laut seine Gedanken zu äußern. Diese Äußerungen werden entweder protokolliert oder aufgenommen und dann später ausgewertet. Vorteil dieser Methode ist die Fülle und Vielfalt der Informationen, welche damit gewonnen werden können, da die Operationalisierung nicht im Vorhinein vorgenommen wird. Der Nachteil ist jedoch gleichzeitig die daraus resultierende Unstrukturiertheit der Daten und die Probleme der Auswertung. Weiterhin besteht die Gefahr, dass aufgrund der tätigkeitsbegleitenden Externalisierung der Gedanken, die eigentliche Interaktion beeinflusst wird, da dadurch manche Prozesse u. U. erst bewusst gemacht werden können. Diese Methode eignet sich aus diesem Grunde vor allem für offene explorative Studien.

Die *Teach Back* Methode [PS72] ist eine hermeneutische Methode um Anwendungswissen zu externalisieren. Teilnehmer werden aufgefordert, einem imaginären Kollegen, meist repräsentiert durch den Versuchsleiter, zu erklären, wie die Aufgabe zu erledigen sei. Zu diesem Zweck kann Papier und Bleistift oder auch andere Werkzeuge genutzt werden. Prozedurales Wissen wird vor allem durch die Fragen der Art *wie kann ich...?* abgerufen, semantisch-konzeptuelles Wissen durch die Fragen der Art *was ist...?*. Vorteil dieser Methode ist die Strukturiertheit anhand der vorgegebenen Aufgabe, nachteilig wirkt sich bei der Beschreibung einer graphischen Anwendung die Tendenz der Teilnehmer aus, sich anhand der graphischen Oberfläche auszudrücken, was natürlich nur eine Ebene [Nor88] der Aufgabe beschreibt.

Die *Beobachtung und Protokollierung* des Benutzerverhaltens und die Analyse der Fehler gibt vor allem Auskunft über die mentalen Konzepte, welche die konkrete Interaktion betreffen. Diese liegen auf der darstellenden und lexikalischen Ebene. Diese Methode ist sehr aufwändig und langwierig, erlaubt aber eine objektive Erfassung des Verhaltens des Benutzers.

Bei der *Antizipationsmethode* wird der Teilnehmer dazu aufgefordert, spekulative Aussagen über die vermutete Systemreaktion auf eine Eingabe zu machen. Dies kann direkt in der Interaktion mit dem System geschehen oder anhand von Bildern der Abbildung. Vor allem bei der zweiten Methode können Aussagen über die Güte des Designs der Anwendungsoberflächen gemacht werden und über die dadurch induzierten mentalen Repräsentationen des Systems.

Eine alternative Methode, die im Rahmen dieser Untersuchung erstmals vorgeschlagen wurde, ist der direkte Zugriff auf die mentale Repräsentation anhand einer *direkten graphischen Externalisierung*. Hier wird der Teilnehmer aufgefordert, sein mentales Bild der Anwendung aufzuzeichnen. Angenommen wird hierbei, dass ein strukturelles Bild in Form einer graphischen Repräsentation beim Umgang mit der Anwendung entsteht, welches, ähnlich einem Flussdiagramm dem Benutzer dabei hilft, die Übersicht über die beteiligten Prozesse zu erhalten und komplexe Informationen leichter zu organisieren [Car03a, SOBD93]. So konnten beispielsweise Ziefle *et al.* [ZAB06, ZB04] zeigen, dass eine adäquate Vorstellung von der Struktur eines Menüs die Interaktion mit einer Anwendung auf einem Mobilgerät signifikant verbessert (siehe auch Abschnitt 3.3).

Vorteil dieser Methode ist der direkte Zugang zu der Repräsentation. Deren Darstellung allerdings hängt stark von der Encodierungsstrategie des Benutzers ab, d. h. auf welche Art und Weise er das Anwendungswissen umgesetzt und gespeichert hat. So ist es zwar erwiesen, dass es eine visuelle Speicherung von Informationen gibt [Fro72], jedoch hängt die Entscheidung der Encodierungsstrategie letztendlich von individuellen und kontextspezifischen Parametern ab [Bes92].

Wie häufig in der Psychologie stellt die Operationalisierung, d. h. die Objektivierung und systematische Ordnung dieser Daten die größte Herausforderung dar. Um eine objektive Analyse durchführen zu können, muss das mentale Modell in eine intermediäre objektive Repräsentation übersetzt werden, ein mentales Modell muss also nachgebildet werden. Der *Pathfinder* Algorithmus [Sch90]



Abbildung 2.2: Ansicht des CardSword Programms zur Visualisierung von Begriffsclustern.

basiert auf der Annahme, dass semantische Ähnlichkeit als graphische Nähe repräsentiert werden kann und dient dazu, auf diese Weise graphische Darstellung semantischer Netzwerke zu erzeugen [VPM03, Hud96]. *CardSword* ist ein Open Source Projekt bei Sourceforge, welches zur Clusterung von gruppierten Begriffen genutzt werden kann (s. Abbildung 2.2). Ein weiteres Werkzeug, welches von IBM entwickelt wurde ist *EZSort*, dieses Programm wurde allerdings von IBM eingestellt und ist lediglich über das Archiv verfügbar. Vorteil dieser Methode ist die Möglichkeit, subjektive Ratings auf objektive Strukturen anhand von Ähnlichkeitsmaßen abzubilden. Damit entsteht die Möglichkeit komplexe Daten zu reduzieren und zu interpretieren. Ein entscheidender Nachteil hingegen liegt in der Ebene der Analyse, welche allein auf der semantischen oder Aufgaben- und Zielebene erfolgt und keinerlei Modellierung der prozeduralen, lexikalischen oder physikalischen Ebene erlaubt.

Eine alternative Methode hierzu wurde in dieser Studie angewandt. Das Verhalten des Benutzers wird direkt in die Aufgabenstruktur übertragen und die Fehler und Abweichungen mit dem zugrundeliegenden Aufgabenmodell in Verbindung gesetzt. Damit sind Aussagen über die lexikalische und syntaktische Ebene des mentalen Modells des Benutzers möglich. Aussagen und Reaktionen auf fehlgeleitete Erwartungen lassen Rückschlüsse auf die Semantik, Ziele und Aufgaben zu, stehen aber damit nicht im Vordergrund. Die wichtigste Eigenschaft dieser Methode ist die Fokussierung auf die beobachtbaren Unterschiede zwischen konzeptuellem Modell und mentalem Modell, welche sich in den Fehlern bei der Ausführung manifestieren. Damit erschließt sich ein unmittelbarer Zugang zu den Aufgabenkonzepten.

2.2.3 Aufgabenanalyse

Die tatsächliche Struktur einer Anwendung wird nach Norman [Nor83] im sog. konzeptuellen Modell dargestellt. Diese Struktur vermittelt sich dem Benutzer über den Umweg der verschiedenen Abstraktionsebenen, die oben beschrieben wurden, und resultiert in einer äußeren Darstellung des Systems, dem Systembild, welches dem Benutzer wiederum dazu dient eine mentale Repräsentation des Systems zu erstellen. Um dieses konzeptuelle Modell des Systems zu erfassen muss das System analysiert und auf den Abstraktionsebenen Nielsens [Nie86] beschrieben werden. Da die Anwendung in diesem Sinne als eine komplexe Aufgabe betrachtet werden kann, eignet sich die Methode der Aufgabenanalyse zur Beschreibung dieser Ebenen.

Verschiedene Methoden zur formalen Analyse der Mensch-Maschine-Interaktion existieren. Diese Methoden unterscheiden sich, durch den Aspekt der Interaktion, welcher bei der Analyse im Vordergrund steht [Haa00, WBG88]. Solche Aspekte können sein, die Übersetzung externer Aufgaben in

Systemaufgaben (*External Internal Task Analysis, ETIT* [Mor83]), Benutzerwissen (*Action Language* [Rei81], *Task-Action Grammar, TAG* [Pay84]), Benutzerhandlung (*Goals, Operators, Methods and Selection Rules, GOMS* [CMN83], *Cognitive Complexity Theory, CTT* [KP85], *User Action Notation, UAN* [HH93]) sowie der Systemoberfläche (*Command Language Grammar, CLG* [Mor81], *Extended Task Action Grammar, ETAG* [Tau88]). Eine ausführliche Übersicht, sowie eine Diskussion der Methoden bezüglich ihrer Bedeutung für die Erfassung und Verbesserung der Benutzbarkeit findet sich bei van Welie [Wel01].

Während diese Methoden komplexe und mächtige Sprachen definieren und damit erlauben, die formale Analyse von Computersystemen auf verschiedenen Detail- und Anwendungsebenen durchzuführen, ist die Anwendbarkeit und Nutzbarkeit dieser formalen Methoden begrenzt. Wie van der Veer und van Welie [VW00] argumentieren, sind diese Methoden meist nur auf einen Ausschnitt der Mensch-Maschine-Interaktion bezogen. Weiterhin sind sie durch die textbasierte und komplexe Syntax schwer zu erlernen und anzuwenden. Die umfassendsten Ansätze, wie die *CLG* und *ETAG* sind zugleich die Methoden, welche die komplexeste Syntax erfordern, und wie de Haan [Haa00] argumentiert, Unterstützung für den Anwender erfordern [HVV91, Haa93].

Graphisch orientierte Methoden zur Aufgabenanalyse stellen eine einfachere und intuitive Alternative zu diesen komplexen textbasierten Methoden dar. Die einfachste Methode ist die *hierarchische Task Analyse* nach Annett und Duncan [AD67], welche einfache hierarchische Bäume als Repräsentation vorschlagen. Hierauf aufbauend entwickelten Paternò *et al.* [PMM97] die *ConcurTaskTree* Notation, welche leicht und intuitiv anzuwenden ist und in verschiedenen modellbasierten Ansätzen zur Entwicklung von Benutzerschnittstellen eingesetzt wird. Mit einem graphischen Editor (CTTE, [MPS02]) steht ein Werkzeug zur Erzeugung dieser Diagramme frei zur Verfügung. Leider ist diese Notation auf die Beschreibung der Arbeitsabläufe beschränkt und bietet nur eingeschränkte Möglichkeiten andere Ebenen zu beschreiben.

Mit *EUTERPE* stellen van Welie und van der Veer [VW00, WVK00, WVE98] eine Umgebung zur Bearbeitung und Evaluierung von Aufgabenmodellen vor, welche auf einer Aufgabenontologie [WVE98] basiert. Verschiedene Aspekte der Interaktion, wie Aufgabe, Objekte, Rollen, Agenten und Ereignisse können hier graphisch spezifiziert werden. *EUTERPE* basiert auf dem Konzept der *User's Virtual Machine (UVM)* [Tau88], welche die Wahrnehmung der Anwendung durch den Nutzer als separates Abbild der Anwendung modelliert und in Funktionalität, Dialog und Präsentation aufteilt. Diese Aufteilung fasst z. T. Ebenen der Mensch-Maschine-Interaktion, wie sie in anderen Modellen beschrieben werden, zusammen und verliert dadurch an Aussagekraft, wird zugleich aber einfacher zu benutzen.

Die Methode des *Contextual Design* [BH98] stellt eine sehr praxisorientierte Methode dar, bei der die Formalismen hinter die Entwicklungsmethodik zurückgestellt wurden. Beyer und Holtzblatt [BH98] analysieren den Arbeitsprozess in je einem Fluss-, Sequenz-, Artefakt-, Kultur- und physikalischen Modell und bieten hierauf aufbauend eine Zuordnung der Ebenen zu Aspekten des zu entwickelnden Systems an.

Die *Unified Modelling Language (UML)* [RJB97] bietet bereits einige graphische Formalismen an, mittels welcher das Ergebnis einer Aufgabenanalyse dargestellt werden kann, wie z. B. das *Activity Diagram*, *Collaboration Diagram*, *Sequence Diagram* oder *Use Case Diagram*. Keine dieser Darstellungen ist allerdings vollkommen ausreichend [SP01]. Verschiedene Anstrengungen wurden bereits unternommen, um eine Aufgabenanalyse in existierende UML-Werkzeuge zu integrieren und die UML-Notation zu erweitern [AHB⁺98, MM00, SP01].

In der Benutzerstudie wurde den Teilnehmern auf beiden Geräten ein fester Ablauf von Aufgaben vorgegeben. Als Anwendung, welche auf beiden Geräten verglichen werden sollte, diente das Kalender-, Adress- und Email-Programm von *Microsoft, Outlook* für den PC und *Pocket Outlook* für die Pocket PC Plattform, eine der verbreitetsten *Personal Information Manager (PIM)* Anwendungen, welche zugleich für verschiedene Plattformen verfügbar ist. Die Aufgaben, welche der Benutzer durchzuführen hatte, entsprechen einem typischen Nutzungsverlauf. Die Instruktion sah für beide Geräte gleich aus:

Bitte führen Sie die Aufgaben gründlich durch und **kommentieren Sie laut** was Sie tun. Probleme, Missverständnisse und Bemerkungen möchten wir Sie bitten, **laut zu äußern**.

Bitte arbeiten Sie die Aufgaben in der **vorgegebenen Reihenfolge** ab und überspringen Sie keine.

Es gibt **keine** falschen oder „dummen“ Kommentare. Uns interessiert Ihr individueller subjektiver Eindruck.

Bitte lösen Sie schrittweise folgende Aufgaben:

1. Schauen Sie bitte nach, ob Sie neue Emails erhalten haben.
2. Schauen Sie bitte nach, ob Sie nächsten Samstag einen Termin haben.
3. Sie wollen nächsten Freitag um 19:00 mit zwei Freunden, Karla und Paul, zu „Bei Nino“ essen gehen. Der Termin ist privat. Sie möchten eine halbe Stunde vor Beginn erinnert werden. Bitte tragen Sie den Termin ein.
4. Sie wollen bei „Nino“ einen Tisch reservieren. Suchen Sie bitte die Telefonnummer von „Bei Nino“ heraus.
5. „Bei Nino“ ist erst ab 20:00 Uhr offen. Sie haben einen Tisch bestellt und Sie wollen Karla und Paul Bescheid sagen, dass sie sich erst um 20:00 Uhr treffen können. Sie wollen den beiden hierzu eine Email schicken, stellen aber fest, dass Karlas Adresse noch gar nicht eingetragen ist.
 - Bitte tragen Sie zunächst Karlas Adresse als neuen Kontakt ein.

Karla Mayer
 Adresse privat: Finkenweg 12, 64283 Darmstadt
 Tel privat: 06151/657687
 Tel mobil: 0172/1231235
 Email: km@maybaum.de
 Geburtstag: 12.11.1972

- Schicken Sie eine Email an Karla und Paul, in der Sie die beiden über die Änderung des Termins informieren.

Um Aussagen über das Nutzerverhalten machen und um Rückschlüsse auf die zugrundeliegenden kognitiven Prozesse ziehen zu können, mussten sowohl Anwendung als auch Aufgaben einer gründlichen Analyse unterzogen werden. Da der Aufbau des Systems und vor allem die Unterschiede des Systems auf den beiden Plattformen als ausschlaggebend für die Unterschiede im Nutzerverhalten angenommen wurden, mussten die möglichen Einflussgrößen erfasst und analysiert werden. Die Unterschiede in den beiden Instanzen des Systems sollen in der Folge mit dem Nutzerverhalten verglichen werden und festgestellt werden ob sich auftretende Probleme auf diese Unterschiede zurückführen lassen. Hieraus sollen sich letztendlich dann die Anforderungen an die plattformübergreifende Entwicklung von Benutzungsoberflächen ableiten lassen.

Drei der in der Studie durchgeführten Aufgaben wurden auf Basis der Abstraktionsebenen von Nielsen [Nie86] analysiert und anhand der *Unified Modeling Language (UML)* [RJB97, SP01] schematisch dargestellt. Die Aufgaben- und Zielebene wurde mit Hilfe eines *Use Case Diagramms* dargestellt (s. Abbildung 2.3). Die Objekte, Attribute und Operationen der Semantischen Ebene wurden mittels Klassendiagramme dargestellt (s. Abbildung 2.4). Der Ablauf der Aufgabe wurde in einem Aufgabenmodell in der Syntax des Aktivitätsdiagramm dargestellt. Eine detaillierte Analyse der einzelnen Interaktionsschritte der Lexikalischen und Physikalischen Ebene wurde nicht formalisiert umgesetzt, sondern in Form von nicht-formalen Anmerkungen in die syntaktische Darstellung integriert, was die Auswertung gegenüber der formalen Ansätze [SP01, MM00] erheblich erleichtert und beschleunigt.

2.2.4 Fehleranalyse

Die Analyse der Fehler, welche die Teilnehmer der Studie bei der Durchführung der Aufgaben machten, soll dazu dienen, Rückschlüsse auf die zugrundeliegenden kognitiven Prozesse zu ermöglichen.

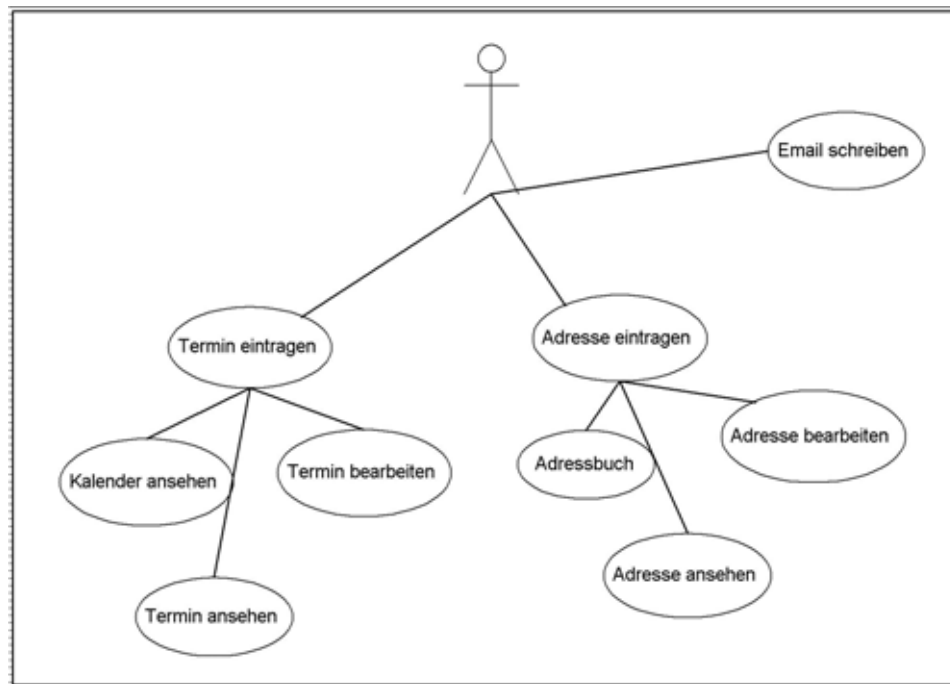


Abbildung 2.3: Use Case Diagramm der Aufgaben in der Benutzerstudie

Die dieser Analyse zugrundeliegende Annahme ist die, dass eine große Ähnlichkeit einer Anwendung auf verschiedenen Endgeräten es ermöglichen sollte, einen großen Teil des Anwendungswissens, welches auf dem einen Gerät erworben wurde, auf dem anderen Gerät wiederzuverwenden. Basierend auf der Fehlertaxonomie von Reason [Rea90] werden die Fehler der Teilnehmer klassifiziert und verschiedenen Kategorien zugeordnet.

Nach Fitt's Theorie des Wissenserwerbes [FP67, And95] erfolgt der Erwerb von Anwendungswissen typischerweise in drei Phasen. In der ersten Phase wird das deklarative Wissen aufgebaut, welches zur Ausführung notwendig ist. In der zweiten Phase wird dieses deklarative Wissen in Regeln oder Prozeduren zusammengefasst und so effektiver geordnet. Die dritte Phase umfasst die Übungsphase und die Optimierung der Handlungsablaufes.

Aufbauend auf dieser Annahme formulierte Rasmussen [Ras86] seine *Skill, Rule und Knowledge (SRK)* Klassifikation der Handlungsebenen. Die drei Begriffe Fertigkeit (*skill*), Regel (*rule*) und Wissen (*knowledge*) beziehen sich hierbei auf den Grad der bewussten Kontrolle, welche bei der Ausführung einer Handlung notwendig ist und auf den Grad der Übung und Expertise des Handelnden.

Bei der wissensbasierten Ausführung benötigt der Handelnde die volle Aufmerksamkeit, um die Informationen, welche ihm zur Verfügung stehen, zu analysieren, die richtigen Schritte einzuleiten und deren Ergebnis zu überprüfen. Dies ist dann der Fall, wenn der Handelnde sich das erste Mal in dieser Situation befindet und noch keine Handlungsrouninen entwickelt hat.

Wurde eine Handlung abgeschlossen, werden Regeln gebildet, welche an Handlungen geknüpft sind. *Wenn ein bestimmter Zustand eintritt, dann verfare wie folgt....* Die Aufmerksamkeit ist hier nicht mehr über die gesamte Handlungsspanne gebunden, sondern wird mit der Auswahl einer Regel von einer halbautomatischen Prozedur abgelöst. Es müssen nicht mehr alle Fakten in Betracht gezogen werden, sondern ein einfacher Muster-Vergleich zwischen Regelbedingung (*wenn*) und relevanten Faktoren reicht aus. Diese regelbasierte Handlungsebene wird mit zunehmender Übung von automatisierten Prozeduren abgelöst.

Auf der fertigkeitbasierten Handlungsebene wird die bewusste Kontrolle weitgehend ausgeschaltet und bekannte Routinen werden abgespielt. Eine auslösende Bedingung führt zur Initiierung der Sequenz.

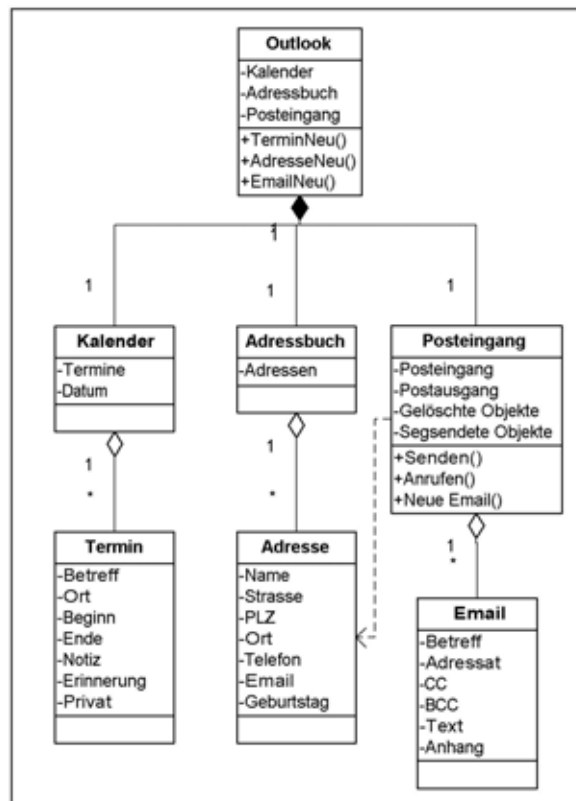


Abbildung 2.4: Objekthierarchie der Anwendung

Mit dem *Generic Error Modelling System (GEMS)* wendet Reason [ER86, Rea90, GSK93] diese Klassifikation der Handlungsebenen auf die Modellierung von Fehlern an. Reason stellt fest, dass als Ursache fehlerhaften Verhaltens dieselben kognitiven Mechanismen verantwortlich sind, welche die nicht-fehlerhaften Verhalten steuern und dass damit fehlerhaftes Verhalten ebenfalls zur Erklärung normalen Verhaltens herangezogen werden kann [Gra00]. Reason versucht die Fehlerkategorien Normans [Nor88], *slips* und *mistakes*, in das Modell der Handlungsebenen von Rasmussen zu integrieren (s. Abbildung 2.5).

“The difference between slips and mistakes are readily apparent in the analysis of the seven stages of action. Form an appropriate goal but mess up in the performance, and you’ve made a slip. Slips are almost always small things: a misplaced action, the wrong thing moved, a desired action undone. [...] Form the wrong goal, and you’ve made a mistake. Mistakes can be major events, and they are difficult or even impossible to detect—after all, the action performed is appropriate for the goal.” [Nor88, S. 106]

Nach Reason sind *slips* Fehler, welche auf der fertigkeitbasierten Handlungsebene auftreten, sog. *skill-based errors*. Wie Norman beschreibt, handelt es sich hier um Fehler, bei welchen die Handlungsintention stimmt, die Durchführung jedoch misslingt. Dies ist nach Reason dann der Fall, wenn in einer Situation die falsche Routine ausgelöst wird. Dies geschieht vor allem dann, wenn das Aufmerksamkeitsniveau niedrig ist und die Handlungen hoch automatisiert ablaufen. Genau dies ist auf der fertigkeitbasierten Ebene der Fall. Besonders leicht können derartige Fehler auftreten, wenn in bekanntem Kontext neue Bedingungen auftreten, die häufigen Bedingungen ähnlich sind, dann werden so genannte *strong but wrong* Fehler auftreten. Landläufig sind diese auch als Gewohnheitsfehler bekannt.

Mistakes hingegen ordnet Reason den beiden anderen Handlungsebenen des SRK Modells zu. Reason unterscheidet in regelbasierte (*rule-based*) und wissensbasierte (*knowledge-based*) Fehler.

Regelbasierte Fehler treten auf, wenn der Handelnde die falsche Regel anwendet um die Situation zu beurteilen. Er beurteilt damit die Situation falsch und entwickelt aus diesem Grunde eine falsche

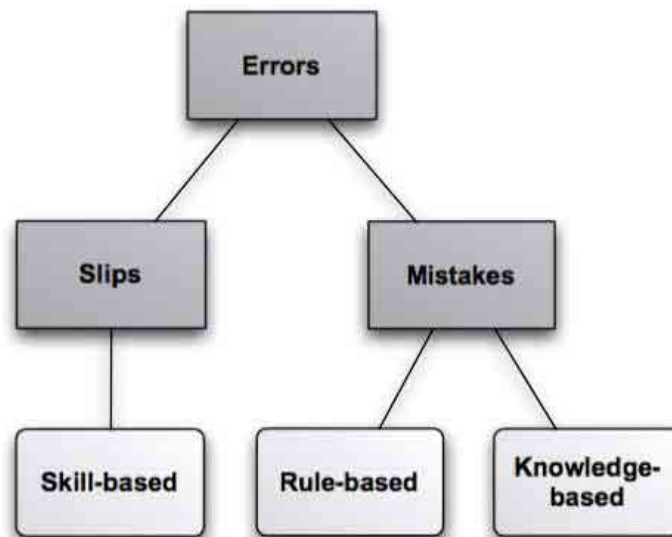


Abbildung 2.5: Fehlertaxonomie von Reason [Rea90]

Handlungsstrategie. Regelbasierte Fehler treten vor allem dann auf, wenn der Handelnde Erfahrung in einer Aufgabe hat, diese aber nicht automatisiert ausführt. Häufig auftretende Regeln haben die Tendenz in ungewöhnlichen Situationen durchzuscheitern und die Beurteilung der Situation zu beeinflussen.

Wissensbasierte Fehler treten dann auf, wenn der Handelnde die Situation aufgrund mangelnder Expertise oder aufgrund zu großer Informationsmenge überfordert ist. In beiden Fällen kann die Situation nicht richtig beurteilt werden, weil sie nicht vollständig berücksichtigt wird. Der Handelnde kennt die Situation also nicht wirklich, ist aber unter Umständen der Meinung er kontrolliere sie. Dies tritt vor allem in neuen und ungewohnten Situationen auf.

Die SRK-Klassifikation von Rasmussen und das darauf aufbauende Fehlermodell Reasons können nun genutzt werden, um die Fehler, welche bei der Benutzung der Anwendung auf den verschiedenen Endgeräten auftreten, zu klassifizieren.

Wissensbasierte Fehler (WBF) treten nach Reason dann auf, wenn sich dem Benutzer die Situation als neu darstellt, da er dann diese auf einer bewussten Ebene der Informationsverarbeitung analysieren muss und daraus Regeln ableiten muss. In der Studie ist das Auftreten dieser Fehler also dann zu erwarten, wenn kein Vorwissen vorhanden ist. Im Fokus unserer Untersuchung steht der Übergang zwischen den Endgeräten und der Transfer von Wissen über die Systemgrenzen hinweg. Bei dem Wechsel zwischen den Endgeräten wird das Auftreten von WBF also dann erwartet werden, wenn kein Wissen von der vorhergehenden Plattform transferiert werden konnte. Typische Fehlerursachen dieser Ebene sind:

- Unterschiede in der menschlichen Kognition.
- Nichtberücksichtigung von Nebeneffekten.
- Verlust des „Roten Fadens“.
- Ausprobieren von alternativen Hypothesen.
- Betonung unwichtiger Ziele.

Regelbasierte Fehler (RBF) sind dann zu erwarten, wenn zwar Regeln (Vorwissen) existieren, diese aber nicht adäquat angewendet werden können. Die Regel scheint ihm u. U. zwar die Passende, ist dennoch nicht zielführend. Dieser Fehlertyp ist dann zu erwarten, wenn der Benutzer unangemessene Regeln auf die neue Situation zu übertragen versucht. In der Testsituation ist dies v. a. dann

zu erwarten, wenn zwischen den Anwendungen auf den verschiedenen Plattformen scheinbare Analogien bestehen, welche den Benutzer zu falschen Schlüssen animieren, im Sinne von, *wenn das vorhin so ging, dann geht das jetzt auch so*. Typische Fehlerursachen auf dieser Ebene sind:

- Vergessen von Aktionen.
- Ziehen falscher Analogien zu bekannten Situationen.
- Nichterkennen der Notwendigkeit einer bewussten Analyse (Wechsel zu wissensbasierter Ebene).

Fertigkeitsbasierte Fehler (FBF) sind nach Reason in hoch automatisierten Situationen zu erwarten. Der Benutzer führt Aktionen aus, die ihm vertraut sind und bei denen er sich nicht mehr konzentrieren muss. In dieser Studie werden FBF dann erwartet, wenn Wissen bereits aus den Vorerfahrungen des Benutzers mit anderen Plattformen mitgebracht wird. Die doch vergleichsweise kurze Dauer des Tests kann vermutlich nicht ausreichen, um derart hoch automatisierte Handlungsmuster zu generieren. Typische Fehlerursachen dieser Handlungsebene sind:

- Motorische Ungenauigkeiten.
- Fehlorientierung.
- Stereotypisches Verhalten.
- Stimulus- / Reaktionsinkompatibilitäten.
- Wahrnehmungsstörungen.

Die Zuordnung und den Ursprung der Fehler, welche in der Benutzerstudie zu erwarten sind, werden schematisch in Abbildung 2.6 dargestellt.

Wie bereits oben beschrieben, werden FBF der gewohnten Plattform und den häufig vom Benutzer verwendeten Anwendungen zugeschrieben. Der hohe Grad der Automatisierung der hierzu notwendig ist, kann aufgrund der kurzen Dauer nicht innerhalb des Experimentes aufgebaut worden sein. RBF wiederum werden als Übertragungsfehler von der ersten Plattform interpretiert. Das neu erworbene Wissen von der Anwendung ist als Regel repräsentiert und wird auf die zweite Plattform angewendet. Natürlich kann in einer neuen Situation die Aufmerksamkeit auch bei automatisierten Handlungen heraufgesetzt sein, so dass auch diese, eigentlich der fertigkeitsbasierten Ebene zugeschriebenen Handlungen, auf einer regelbasierten Ebene durchgeführt werden können.

Die letzte Fehlerart sind die WBF, welche darauf schließen lassen, dass zuvor angeeignetes Wissen nicht angewendet werden kann. FBF entspringen also dem Transfer von Vorwissen aus der ansonsten verwendeten Plattform, RBF aus dem Transfer von Wissen aus der ersten Plattform und KBF entstehen aus Fehlern bei der Benutzung und Aneignung der Regeln für die zweite Plattform.

2.2.5 Methode

Die Benutzerstudie wurde mit acht Teilnehmern durchgeführt die sich auf eine Email-Anfrage hin gemeldet hatten. Jede Person erhielt für ihre Teilnahme ein kleines Sachgeschenk im Wert von ca. 10 Euro. Zwei der Teilnehmer waren aus der Altersgruppe der 20 bis 29-jährigen, sechs der Teilnehmer waren zwischen 30 und 39 Jahren. Alle Teilnehmer hatten ein abgeschlossenes Studium und mittlere bis hohe Erfahrung mit Desktop PC. Alle Teilnehmer hatten keine bis geringe Erfahrungen mit *Microsoft Outlook*.

Die Teilnehmer wurden ihren Vorkenntnissen nach in zwei Experimentalgruppen eingeteilt. Die erste Gruppe, bestehend aus fünf Personen, arbeitete im Alltag vorwiegend auf Desktop PCs mit dem Windows Betriebssystem von Microsoft und hatte keine Vorerfahrungen mit Mobilgeräten. Diese Gruppe wird in der Folge als die PC-Gruppe (PCG, $n = 5$) bezeichnet. Die zweite Gruppe, die Handheld Gruppe (HHG, $n = 3$ Personen), bestand aus Teilnehmern, die im Alltag mit Computern arbeiteten, auf denen andere Betriebssysteme (Mac OS, Linux, BeOS) installiert waren. Diese Teilnehmer besaßen mittlere bis große Erfahrung im Umgang mit Handheld-Geräten, nicht aber dem PocketPC-System sondern Geräten mit *Palm* oder *Symbian OS*.

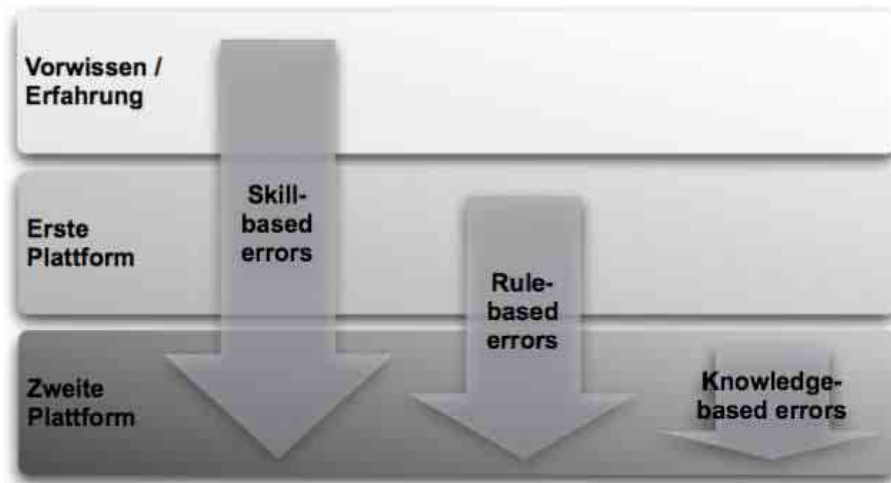


Abbildung 2.6: Zusammenhang zwischen Art der Fehler und Ursprung des zugrundeliegenden Wissens.

Aufgrund der Tatsache, dass Teilnehmer, welche zwar Erfahrungen auf dem PocketPC, jedoch keine Erfahrung mit Windows PCs haben, nicht zu finden waren, war es nicht möglich, die perfekte Gegengruppe zur ersten Experimentalgruppe zu gewinnen. Die alternative Vorgehensweise, MacOS, Linux und BeOS Benutzer mit Vorerfahrungen auf anderen mobilen Plattformen zu rekrutieren, erschien als eine ausreichende Kontrastierung geeignet.

Die PC-Gruppe sollte nun zuerst die Aufgaben auf der, ihr bekannteren Plattform, also dem Windows PC durchführen und dann auf den PocketPC wechseln. Bei der Handheld-Gruppe hingegen sollte zunächst die Aufgabe auf der PocketPC Plattform durchgeführt werden, um dann auf den Windows Desktop zu wechseln. In beiden Gruppen sollte also zunächst die vertrautere Plattform genutzt werden, um dann auf die neue unbekannte Plattform zu wechseln. Tabelle 2.1 zeigt den Ablauf der Studie. Ziel war es, Transfer-Effekte möglichst klar zu isolieren.

Der Versuch wurde durchgeführt auf einem *Shuttle PC* mit AMD CPU und einem 17" LG Flüssigkristall Monitor. Als Eingabegeräte wurde eine Funk-Maus und Tastatur von *Logitech* verwendet. Als Handheld Gerät kam ein *Compaq Ipaq 3870* mit ARM SA1110-Prozessor und Farbdisplay mit 240x320 Pixel Auflösung zum Einsatz. Auf dem *Shuttle PC* lief *Microsoft Windows 2000 Professional* und *Microsoft Outlook 2002*, auf dem *Ipaq* war *Microsoft PocketPC 3.0* mit *Microsoft Pocket Outlook* installiert.

Aufgrund der durch die gezielten Anwerbung der Teilnehmer geringe Versuchsgruppe konnten Bedingungen und Teilnehmer nicht randomisiert werden. Da dies für ein Experiment notwendige Voraussetzungen sind [CS66], handelt es sich bei dieser Studie um eine Feldstudie. Als experimentelles Design wurde also eine Feldstudie mit zwei Gruppen und einem Prä-/Posttest Vergleich gewählt. Unabhängige Variablen waren PC-Erfahrung und Erfahrung mit Handheld-Geräten sowie die Richtung des Wechsels zwischen den Plattformen. Auf ein gekreuztes Design wurde anhand des daraus entstehenden Umfangs und anhand des Charakters als explorative Feldstudie verzichtet.

Als abhängige Variablen wurden folgende Maße erhoben:

- Die *Benutzerzufriedenheit* wurde nach jedem Durchgang mit Hilfe der *Overall User Reactions* Skala des *Questionnaire for User Interaction Satisfaction (QUIS)* [CDN88] erhoben. Die Skalenelemente wurde aus [Shn98, S.135 ff.] entnommen und ins Deutsche übersetzt. Die Elemente wurden in eine siebenstufige Likert-Skala überführt und zur computergestützten Erhebung in C# implementiert. Die Skalenelemente waren: „Mein Eindruck von der Anwendung war...“, wunderbar ↔ furchtbar, befriedigend ↔ frustrierend, stimulierend ↔ langweilig, leicht ↔ schwierig, angemessener ↔ unangemessener Funktionsumfang, flexibel ↔ starr. Die Daten haben ordinales Skalenniveau. Die Einzelwerte wurden gerichtet zu einem Summenwert addiert.

	PC-Gruppe (n=5)	Handheld-Gruppe (n=3)
1	Instruktion, Datenschutzvereinbarung und Fragen zur Person	
2	Einweisung in den PocketPC	
3	Microsoft Outlook auf Windows PC	Microsoft Pocket Outlook auf PocketPC
4	Fragebogen zur Zufriedenheit	
5	Fragebogen zur Belastung	
6	Erfassung mentales Modell (Teach Back)	
7	Direkte Graphische Externalisierung	
8	Antizipationsmethode mit Screenshots v. Pocket Outlook	Antizipationsmethode mit Screenshots v. Outlook
	Pause	
9	Microsoft Pocket Outlook auf PocketPC	Microsoft Outlook auf Windows PC
10	Fragebogen zur Zufriedenheit	
11	Fragebogen zur Belastung	
12	Erfassung mentales Modell (Teach Back)	
13	Direkte Graphische Externalisierung	
14	Bemerkungen und Dank	
	Gesamtdauer ca. 1:30h	

Tabelle 2.1: Ablauf der Studie

- Die *Belastung* der Benutzer bei der Aufgabendurchführung wurde ebenfalls nach jedem Durchgang erhoben. Hierzu wurde eine Übersetzung des *NASA Task Load Index (TLX)* [HS88] angefertigt und in C# implementiert.
- Eine *Videoaufzeichnung* wurden während der Durchführung angelegt. Diese wurde nach verschiedenen Kriterien ausgewertet.
 - Äußerungen
 - Fehler bzw. Abweichungen
- Erfassung des *Mentalen Modells*
 - Teach Back Methode
 - Direkte Graphische Externalisierung
 - Antizipationsmethode

2.2.6 Benutzerbelastung und Zufriedenheit

Die Auswertung des Fragebogens zur Benutzerzufriedenheit (QUIS) erfolgte auf Basis der siebenstufigen Ratings in sechs Skalen (s. o.). Die ordinal-skalierten Werte wurden für jede Person zu einem Summenwert verrechnet. Die Überprüfung der Nullhypothese, dass beide Plattformen zu derselben Benutzerzufriedenheit führen ($H_0 : Zufriedenheit_{PC} = Zufriedenheit_{PocketPC}$) gegen die Alternativhypothese ($H_1 : Zufriedenheit_{PC} > Zufriedenheit_{PocketPC}$), wurde für diese Summenwerte mit Hilfe des nichtparametrischen *Wilcoxon Signed Ranks Test* überprüft

[Wil45, Bor89]. Erwartungsgemäß konnte die Nullhypothese mit über 95% Wahrscheinlichkeit verworfen werden, da die Benutzerzufriedenheit bei der Benutzung des PC signifikant höher war als bei der Benutzung des mobilen Gerätes ($p[Z = -2,527] = .012$; $\sigma_{PC} = 3,381$; $\sigma_{PocketPC} = 4,175$).

Die Auswertung des *NASA Task Load Index (TLX)* mit den ebenfalls siebenstufigen Ratings auf sieben Skalen erfolgte ebenfalls auf Basis der Summenwerte. Eine Gewichtung der Skalenwerte fand nicht statt. Auch hier war die Nullhypothese die Annahme, dass sich beide Plattformen nicht in ihrer Belastung unterscheiden ($H_0 : Belastung_{PC} = Belastung_{PocketPC}$) gegenüber der Alternativhypothese, derzufolge die Belastung auf dem mobilen Gerät höher sei ($H_1 : Belastung_{PC} < Belastung_{PocketPC}$). Auch hier fand die statistische Auswertung der Mittelwertsunterschiede der ordinalskalierten Werte mit Hilfe des *Wilcoxon Signed Rank Tests* statt. Auch bei den Belastungswerten zeigten sich die Ergebnisse erwartungskonform, denn auch hier schnitt die mobile Plattform schlechter ab, da sie zu höheren Belastungswerten führte ($p[Z = -1,992] = .046$; $\sigma_{PC} = 5,849$; $\sigma_{PocketPC} = 7,924$).

Die Wechselwirkungen zwischen Gruppe, Plattform und der Zufriedenheits- bzw. Belastungsbeurteilung waren in beiden Fällen nicht signifikant. Die Unterschiede in der Zufriedenheit zwischen den Geräten zeigten sich unabhängig davon, welches Gerät der Teilnehmer zuerst benutzte.

Bei der Belastung zeigte sich eine Tendenz dahingehend, dass die Unterschiede in der Beurteilung für die Teilnehmer der HHG (PocketPC \rightarrow PC) weniger stark zeigten, als bei den Teilnehmern der PCG. Diese Wechselwirkung ist jedoch nicht signifikant (Lineare Regression [Fis22]: $p_{Gerat}[t = -2,616] = .021$; $p_{Gruppe}[t = 0,48] = .833$). Eine mögliche Erklärung für diese Tendenz kann in Reihfolgeeffekten liegen, welche die Belastungswerte überlagern (bei HHG war der PC das zweite Gerät, und damit kann hier bereits ein Ermüdungseffekt aufgetreten sein, wohingegen die PCG den PocketPC als zweites Gerät erhielt und damit die negative Bewertung den wahren Wert überschätzt).

Zusammenfassend lässt sich feststellen, dass die quantitativen Ergebnisse der Benutzerzufriedenheits- und Belastungserhebung erwartungskonform sind. Das mobile Gerät wird generell als weniger zufriedenstellend und anstrengender bewertet. Gründe hierfür liegen zweifellos in der weniger gewohnten Handhabung aber auch in der umständlichen Eingabe über die Tastatur (ein Punkt der sich auch deutlich in den Videodaten findet).

Wechselwirkungen in der Reihenfolge (wird ein Gerät als angenehmer oder einfacher empfunden, wenn zuvor das andere verwendet wurde) ließen sich nicht nachweisen. Dies lässt auch darauf schließen, dass die Ergebnisse der anderen Methoden auf die Unterschiede zwischen den Geräten und nicht auf Reihenfolgeeffekte zurückzuführen sind.

Die Interpretierbarkeit der statistischen Kennwerte ist freilich nur bedingt gegeben, da es sich bei dieser Untersuchung zum einen um eine Feldstudie handelt und damit eine strenge Kontrolle der Einflussgrößen nicht gegeben ist, und zum anderen die Größe der Versuchsgruppe kaum ausreicht, um valide Aussagen treffen zu können [Len01]. Dies wird im Rahmen dieser explorativen Studie jedoch nicht als notwendig erachtet, da hier die qualitativen Daten, welche im folgenden diskutiert werden, im Vordergrund stehen und die vorliegenden quantitativen Ergebnisse, zumal erwartungskonform, nur der Vollständigkeit halber Erwähnung finden.

2.2.7 Auswertung der Videodaten

Die Auswertung der Videodaten erfolgte in mehreren Schritten. Zunächst wurden für jeden Teilnehmer und jede Bedingung die Durchführung protokolliert und den einzelnen Aufgabenphasen zugeordnet. Diese Phasen standen als Ergebnis der Aufgabenanalyse (s. Abschnitt 2.2.3) zur Verfügung. Abweichungen vom ideale Durchführungspfad (Umwege, Auslassungen, etc.) wurden als Fehler gewertet und entsprechend der Klassifikation (s. Abschnitt 2.2.4) eingeordnet. Weiterhin wurde versucht, die Fehler den Handlungsebenen Nielsens [Nie86, Mor81] zuzuordnen (s. Tabelle 2.2). Eine quantitative Auswertung der Fehler erfolgte nicht. Die Probanden werden in der Folge einheitlich als Teilnehmer bezeichnet ungeachtet des tatsächlichen Geschlechts.

2.2.7.1 Klassifikation der Fehler

Handlungsebene	Wissensbasierte Fehler	Regelbasierte Fehler	Fertigkeitsbasierte Fehler
Aufgabenebene			
Zielebene			
Semantische Ebene			
Syntaktische Ebene			
Lexikalische Ebene			
Physikalische Ebene			

Tabelle 2.2: Klassifikationsmatrix der Benutzerfehler

Beispiele für wissensbasierte Fehler: Wissensbasierte Fehler lassen sich nach der Fehlertaxonomie meist auf die mangelhafte Benutzbarkeit der aktuellen Plattform zurückführen. Diese Fehler treten dann auf, wenn der Benutzer nicht in der Lage ist, die Situation korrekt einzuschätzen und aufgrund des verfügbaren Wissens und der verfügbaren Informationen keine richtige Entscheidung treffen kann. Diese Fehler sind demnach weniger relevant, betrachtet man die plattformübergreifende Benutzbarkeit. Allerdings kann der Aufbau einer korrekten mentalen Repräsentation auf der einen Plattform bei der Benutzung der zweiten Plattform von Nutzen sein. Die folgenden Beispiele geben eine beispielhafte Übersicht über wissensbasierte Fehler auf verschiedenen Handlungsebenen.

Die Beispiele sind in manchen Fällen auf bestimmte Oberflächenelemente in den Anwendungen bezogen. Da eine ausführliche Schilderung aller beobachteten Fehler den Rahmen dieser Arbeit sprengen würde, soll in diesem Abschnitt nur eine Übersicht über die beobachteten Fehler gegeben werden. Einige besonders einprägsame Beispiele regel- und fertigkeitstbasierter Fehler werden im nachfolgenden Abschnitt etwas ausführlicher geschildert.

- *Vertraulichkeit:* Die meisten Teilnehmer stellten die Vertraulichkeit des Termins nicht auf privat, obwohl dies in der Aufgabenstellung vorgegeben war (Aufgaben- / Zielebene).
- *Nachziehen der Verschiebung:* Nur zwei Teilnehmer verschoben den Termin in ihrem Kalender, nachdem der Anruf bei „Nino“ ergeben hatte, dass dieser erst um 20:00 Uhr stattfinden würde (Zielebene).
- *Analogie zu Papierkalender:* Zwei Teilnehmer zeigten bei der Durchführung, dass ihr mentales Modell in Analogie mit einem konventionellen Papierkalender stand. Zum einen wurde die Tagesansicht als Monatsansicht wahrgenommen, wie dies in vielen Papierkalendern der Fall ist, zum anderen wurde der Termin direkt in die Tagesansicht eingetragen und die zusätzlichen Einstellungen über die Eingabemaske erst nach einigem Suchen entdeckt. Ein Teilnehmer formulierte hierauf die Regel: „erst den neuen Termin anlegen und dann im Dialog einstellen“ und wendete diese dann bei dem zweiten Gerät auch an (Semantische Ebene).
- *Erinnerungszeit:* Viele Teilnehmer hatten Probleme festzustellen, ob die Erinnerungszeit die Erinnerung im Voraus (n Minuten vor Termin) oder die Dauer der Erinnerung (für n Minuten) bezeichnete (Syntaktische Ebene).
- *Vertraulichkeit, Kategorie und Beschriftung:* Die Unterscheidung zwischen Vertraulichkeit, d. h. die Sichtbarkeit bei gemeinsam genutzten Kalendern (privat/öffentlich), Kategorie, d. h. die Einordnung in inhaltliche Sparten (geschäftlich, privat, ...) und Beschriftung, d. h. die Einfärbung in der Kalenderansicht (privat, Besprechung, wichtig, ...) wurde den Teilnehmern meist nicht klar und führte zu Verwirrung (Lexikalische Ebene).

Beispiele für regelbasierte Fehler: Regelbasierte Fehler geben Aufschluss über Konzepte, welche der Benutzer sich im Umgang mit der Anwendung auf der einen Plattform angeeignet hat und welche auf der zweiten Plattform nicht anwendbar sind oder zu fehlerhaften Ergebnissen führen. Auf der anderen Seite können Fehler, die auf der ersten Plattform aufgetreten sind, zur Bildung von Regeln führen, welche zu einer korrekten Erfüllung der Aufgabe auf der zweiten Plattform führen können. Der folgende Abschnitt gibt eine beispielhafte Übersicht über solche Fehler.

- *Posteingang*: Die Regel „Email wird über ein Email-Programm versendet“ wird durch die Bezeichnung „Posteingang“ für die Email-Funktion von Outlook verletzt. Tatsächlich hatten die meisten Benutzer Probleme diese Funktion dem „Posteingang“ zuzuordnen (Zielebene).
- *Dropdown-Listen vs. Eingabemasken*: Während auf dem PC die Auswahl von verschiedenen Feldbezeichnungen über sog. Dropdownlisten geregelt wurde, stand ein sehr ähnliches Symbol beim Handheld Computer für den Aufruf einer Eingabemaske, dies wurde in der Konsequenz von vielen Benutzern verwechselt. Eine fehlerhafte Regel hatte sich etabliert (Semantische Ebene).
- *Orts- und Betreffsfeld*: Bei der Eingabe von Terminen wurde zunächst der Ort zusammen mit dem Betreff eingegeben. Das getrennte Feld für den Ort wurde meist danach wahrgenommen und dann korrigiert. Bei der zweiten Plattform wurde dieser Fehler meist nicht mehr gemacht. Eine neue korrekte Regel hatte sich etabliert (Syntaktische Ebene).
- *Schließen eines Unterdialogs*: Auf dem PC werden Eingabemasken meist mit der Bestätigung über „OK“ geschlossen. Der PocketPC bietet zum Teil Unterdialoge, bzw. Eingabemasken an, die nicht über „OK“ sondern über einen Tipp neben die Maske geschlossen werden. Zugleich ist aber auch der „OK“-Button des Hauptdialogs sichtbar. Eine versehentliche Nutzung des „OK“-Buttons war häufig (Syntaktische / lexikalische Ebene).
- *OK vs. Schließen*: Auf dem PocketPC ist der „OK“-Button oben rechts angebracht, dort, wo beim PC das Kreuz zum Schließen des Fensters ist. Einige Benutzer verwendeten diese Schließen-Funktion nach der Benutzung des PocketPCs auf dem PC, andere suchten eine „OK“-Schaltfläche auf dem PocketPC (Syntaktische / Lexikalische Ebene).
- *Zentrale Funktionsauswahl*: Auf dem PC steht mit der Outlookleiste ein Zugriff auf alle Funktionen zur Verfügung, der PocketPC erlaubt dies über den „Desktop“, allerdings nicht auf die Kontakte. Das Symbol, welches dem Kontakte-Symbol sehr ähnelt steht hier für die Besitzerinformationen. Der Zugang zum Adressbuch wurde in einigen Fällen über dieses Feld gesucht (Lexikalische Ebene).

Beispiele für fertigkeitsbasierte Fehler: Fertigkeitsbasierte Fehler sind Fehler, welche auf hoch-automatisierte und häufig genutzte Prozesse zurückzuführen sind. Diese Abläufe sind also in der Regel Teil der Vorkenntnisse des Benutzers und Teil des Repertoires auf der üblicherweise genutzten Plattform. Die folgende Übersicht zeigt Beispiele für fertigkeitsbasierte Fehler, welche in der Untersuchung beobachtet wurden.

- *Zurück*: Aus verschiedenen Anwendungen (z.B. Internet-Browser) ist die „Zurück“ Funktion bekannt. Auf dem PC steht eine derartige Funktion ebenfalls zur Verfügung. Sie bringt den Benutzer zu der zuletzt verwendeten Ansicht. Weiterhin gibt es eine Schaltfläche für Benachbarte Elemente in einer Liste (z.B. vorhergehender Termin). Zwei Teilnehmer verwendeten diese Funktionen in anderer Weise. Ein Teilnehmer erwartete dass „Zurück“ das Navigieren im Kalender um einen Tag zurück erlauben sollte. Eine anderer Teilnehmer nutzte die Funktion „Vorhergehendes Element“ konsistent als „Zurück“ Funktion (Semantische Ebene).
- *Doppelklick auf Outlookleiste*: Einige Teilnehmer versuchten durch Doppelklick auf das Email-Symbol in der Outlookleiste eine neue Email zu erstellen, wobei diese Leiste nur die Funktion der Navigation zwischen den Outlook-Anwendungen hat (Syntaktische Ebene).
- *Menüaktion*: Im Menü des PocketPC kann ein Menüpunkt auf der obersten Ebene direkt eine Aktion auslösen, anstatt ein Untermenü zu öffnen, wie das bei der PC-Plattform üblich ist. Auch innerhalb der PocketPC-Plattform ist dies nicht einheitlich gestaltet. Teilnehmer waren z.T. erstaunt, wenn die Auswahl des Menüpunktes „Neu“ direkt ein neues Element erzeugte. Ein Teilnehmer interpretierte den Tooltip-Text als Untermenü und versuchte diesen auszuwählen (Syntaktische Ebene).
- *Return*: Die Return-Taste wird meist zur Bestätigung einer Eingabe in einem Eingabefeld verwendet. Bei der PC-Plattform kann man so zwischen den einzelnen Eingabefeldern wechseln, bei der PocketPC Plattform wird mit Return in manchen Fällen der komplette Dialog geschlossen, in anderen nicht (Lexikalische Ebene).
- *Überschreiben*: Es ist allgemein üblich, dass neu eingefügter Text oder Elemente alte überschreiben, wenn dieser alte Text markiert ist. Dieses Vorgehen funktionierte auf dem PocketPC nicht. Ähnlich wenig erwartungskonform war das Verhalten beim Einfügen von mehreren selektierten Einträgen aus dem Adressbuch in das Email-Adressfeld, da hier nur ein Eintrag übernommen wurde (Lexikalische Ebene).

2.2.7.2 Detaillierte Analyse

Wie bereits festgestellt wurde, liegt der Fokus der Betrachtung hier auf den fertigkeiten- und den regelbasierten Fehlern, da diese in starkem Maße auf die Interaktionen zwischen den Plattformen zurückzuführen sind. Wissensbasierte Fehler hingegen werden in erster Linie Rückschlüsse auf die Benutzbarkeit des Systems erlauben. Da dies jedoch nicht Gegenstand unserer Betrachtungen ist, werden im folgenden Abschnitt einige der Fehler detaillierter besprochen, die besondere Einblicke in die Wechselwirkungen zwischen den Anwendungen auf verschiedenen Plattformen bieten.

Dropdown-Liste: Die Verwendung eines nach unten zeigenden Pfeil-Symbols wird auf beiden Plattformen in widersprüchlicher Weise verwendet. Abbildung 2.7 links zeigt den Dialog zur Erstellung eines neuen Kontakts auf dem PocketPC. In den Feldern *Name* und *Adr. Büro* ist auf der rechten Seite ein Pfeilsymbol erkennbar. Die rechte Seite der Abbildung zeigt die Eingabemaske für die Postanschrift, die erscheint, wenn man auf diesen Pfeil tippt.

Abbildung 2.8 zeigt den entsprechenden Dialog auf dem Desktop PC. Auch hier erscheint das Pfeil-Symbol. Hier öffnet der Klick auf den Pfeil eine Auswahlliste auf der man den Namen des Feldes auswählen kann, welches im dazugehörigen Textfeld anzuzeigen ist.

Eine fehlerhafte Übertragung der Regel wie dieses Symbol zu verwenden ist, lässt sich anhand der Videodaten in beide Richtungen nachweisen. Ein Teilnehmer der HHG (d. h. PocketPC → PC) erwartete, nachdem er festgestellt hatte, dass er mit dem Pfeil auf dem PocketPC die Eingabemaske öffnen konnte, dasselbe auf dem PC um die Adresse einzutragen. Er war erstaunt, als sich die Liste der Feldbezeichnungen öffnete. In die umgekehrte Richtung zogen drei Teilnehmer diesen irreführenden Schluss und versuchten auf dem PocketPC den Adresstyp von *geschäftlich* auf *privat* umzustellen und gelangten in die Eingabemaske für die Adresse. Dieses Problem stellte sich als besonders gravierend dar, da, wie im nächsten Abschnitt dargestellt wird, die Teilnehmer häufig versuchten, die Eingabemaske auf dem PocketPC mit *OK* zu schließen und damit der gesamte Kontakt-Dialog geschlossen wurde.

Die Verwendung der OK-Schaltfläche: Das Vorgehen auf beiden Plattformen unterscheidet sich ebenfalls bezüglich des Schließens von Dialogen bzw. bezüglich der Bestätigung und des Speichern von Änderungen. Wie 2.7 zeigt, wird auf der PocketPC Plattform ein Dialog oben rechts mit *OK* geschlossen. Die *OK*-Schaltfläche oben rechts entspricht funktional der *OK*-Schaltfläche in einem PC-Dialog. Dort ist diese meist unten rechts oder in der Mitte platziert und dient zur Bestätigung von Änderungen. Von der räumlichen Platzierung hingegen entspricht dieses *OK* allerdings vielmehr dem *Schließen*-Kreuz des PC (s. Abbildung 2.8).

Tatsächlich zeigte sich bei einigen Teilnehmern der HHG die Tendenz, dieses Vorgehen auf die PC-Plattform zu übertragen und z.B. Dialoge zur Kontakt- oder Termin-Eingabe über das Kreuz oben rechts zu schließen anstatt die dafür vorgesehene Schaltfläche *Speichern und Schließen* in der Symbol-Leiste zu verwenden.

Umgekehrt suchten andere Teilnehmer nach der *OK*-Schaltfläche um einen Dialog zu bestätigen, bzw. versuchten nach einigem Zögern mit *OK* den Dialog zu schließen in der Erwartung die eingegebenen Daten zu verlieren *Mal schauen was da passiert...*

Hat der Teilnehmer dann die Verwendung der *OK*-Schaltfläche akzeptiert, besteht die Gefahr, dass er sie in einem anderen Zusammenhang fälschlicherweise nutzt und damit unvermittelt den aktuellen Arbeitskontext verlässt.

Abbildung 2.9 zeigt eine Eingabemaske zur Eingabe einer Adresse auf dem PC, man schließt diese, indem man die *OK*-Schaltfläche benutzt. Vergleicht man dies mit der äquivalenten Eingabemaske in Abbildung 2.7 rechts, liegt es nahe, dass man auch dort mit *OK* bestätigt. Leider ist dies nicht der Fall, da die Eingabemaske geschlossen wird indem man in den Bereich über oder unter der Eingabemaske tippt. Das Betätigen der *OK*-Schaltfläche schließt dagegen die gesamte Kontakteingabe ab. Der Benutzer verliert zwar nicht die eingegebenen Daten, muss aber den Dialog über mehrere Schritte wieder aufnehmen. Dieser Fehler trat sehr häufig in der PCG bei der Erstellung des Kontaktes oder der Email auf.



Abbildung 2.7: Dialog zur Erstellung eines Kontakts auf der PocketPC Plattform. Auf der linken Seite sind die Pfeil-Symbole erkennbar, über welche die Eingabemasken (s. rechte Seite) zu öffnen sind.

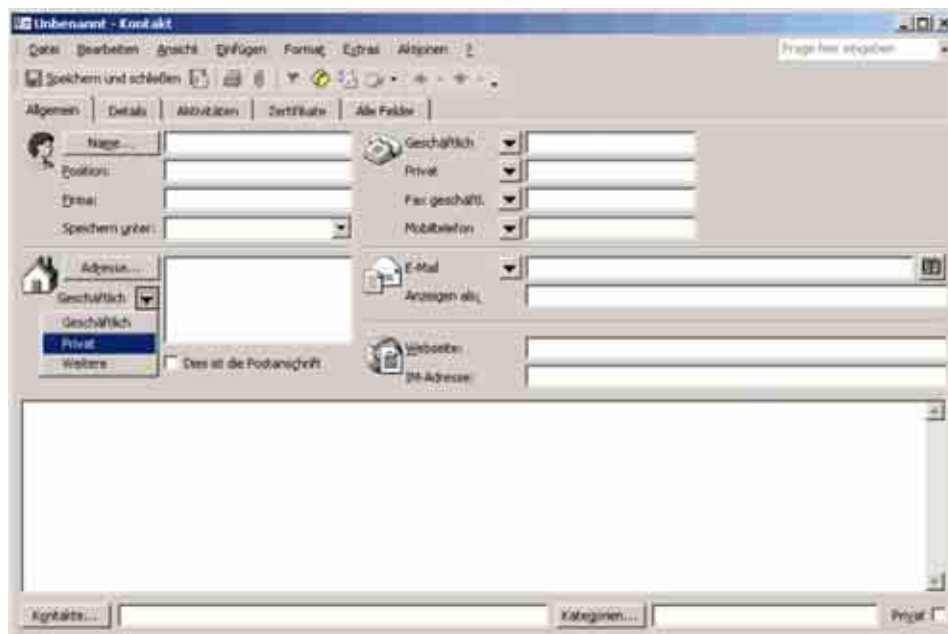


Abbildung 2.8: Dialog zur Erstellung eines Kontakts auf dem PC. Über das Pfeil-Symbol lässt sich das Feld wählen, welches im dazugehörigen Textfeld angezeigt werden soll.

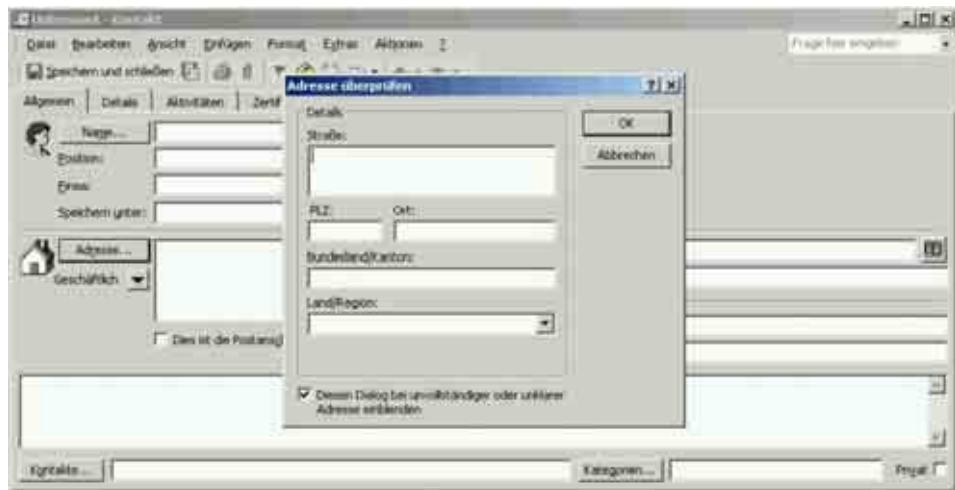


Abbildung 2.9: Adresseingabemaske auf PC

Tastaturbefehle: Ein weiteres Beispiel der inkonsistenten Verwendung von Eingabemetaphern findet sich bei der Umsetzung der Tastatureingaben, welche auf dem PocketPC über das kleine ausklappbare virtuelle Keyboard (s. Abbildung 2.7) realisiert wird.

Im Allgemeinen entsprechen die Funktionen der Tasten dieses virtuellen Keyboards den Funktionen eines üblichen Standardkeyboards. Somit wäre zu erwarten, dass die Verwendung einer entsprechenden Taste auf beiden Plattformen zu demselben Resultat führen sollte. Dass dies nicht durchgehend der Fall ist, stellte sich in der Benutzerstudie in verschiedenen Fällen heraus. In einigen Situationen stellte dies eine ernstzunehmende Fehlerquelle dar.

Auf der PC Plattform kann der Benutzer in einer Eingabemaske, wie z.B. in Abbildung 2.8 dargestellt, eine Eingabe in ein Feld mit der *Return*-Taste bestätigen und in das nächste Feld wechseln. In manchen Kontexten geschieht allerdings schlicht nichts. Auf der PocketPC Plattform hingegen führt die Betätigung der *Return*-Taste in einem Eingabedialog in manchen Fällen zum Schließen dieses gesamten Dialoges. Dieser Fehler trat besonders häufig in der PCG, bei der Eingabe eines neuen Kontakts auf.

Andere kombinierte Tastaturbefehle, wie z. B. Kombinationen mit *Strg* bzw. *Umschalt* sind zum Teil realisiert. Aber auch hier ist die Umsetzung auf dem PocketPC nicht durchgängig konsistent mit der Konvention der PC-Welt. Vor allem zwei Teilnehmer favorisierten diese Tastatureingaben und setzten sie für Aktionen wie *Copy & Paste*, bzw. die mehrfache Markierung in Listen ein.

Auch hier führte dieses Vorgehen nicht in allen Fällen zum Erfolg. So versuchte ein Teilnehmer mehrere Einträge aus der Email-Adressliste zu markieren und in das Adressfeld zu übertragen. Die Übernahme erfolgte jedoch nur für den ersten selektierten Eintrag. Offensichtlich handelt es sich in beiden Fällen um eine fehlerhafte Übertragung von automatisierten Prozessen auf der lexikalischen Ebene.

Das Menü: Der PocketPC hat üblicherweise sein Menü am unteren, anstatt wie der PC, am oberen Rande des Fensters. Diese Designentscheidung hängt u. a. mit der Hauptinteraktionsrichtung zusammen, welche von links oben nach rechts unten verläuft, und mit der besseren Zugänglichkeit durch die Hand und Stift von unten zusammen [Mic04]. Das Menü des PocketPC kann neben klassischen Menüs mit Untermenü auch Schaltflächen mit Icons enthalten. Eine Besonderheit des PocketPC sind Menüeinträge, die wie Schaltflächen direkt Funktionen aufrufen.

Während der gesamten Studie fiel auf, dass die meisten Teilnehmer, beider Gruppen die Menüfunktionen überhaupt nicht oder erst sehr spät wahrnahmen. Das *Neu*-Menü, welches stets im linken unteren Eck erscheint, wurde von fast allen Teilnehmern die meiste Zeit ignoriert. Es stellt sich also generell die Frage, ob diese Designentscheidung zielführend ist.



Abbildung 2.10: Das Menü des Adressbuches auf dem PocketPC. Die Auswahl des Menüs führt direkt zur Erstellung eines neuen Kontakts anstatt ein Untermenü zu öffnen. Die rechte Seite zeigt einen Benutzer, der den Tooltip des Menüpunktes als Untermenü interpretiert und versucht diesen auszuwählen.

Bei allen Teilnehmern führte die Tatsache, dass ein Menüpunkt direkt eine Funktion aufruft, zu einer überraschten, teils erschrockenen Reaktion. Die Auswahl des *Neu*-Menüs führt im Adressbuch zur Erstellung eines neuen Kontakts, wohingegen das Menü mit derselben Bezeichnung in der Startansicht ein Untermenü mit den verschiedenen Inhaltstypen (Email, Kontakt, Aufgabe, etc.) zur Neuanlage öffnet.

Ein weiterer Beleg für die Problematik dieser Inkonsistenz zeigte sich bei einem Teilnehmer. Er interpretierte fälschlicherweise den Tooltip, der sich bei Auswahl des Menüpunktes öffnete, als Untermenü und versuchte ihn auszuwählen (s. Abbildung 2.10). Dieser Teilnehmer versuchte mehrere Male zuerst *Neu* anzutippen. Er blieb mit dem Stift auf dem Menü in Erwartung des Untermenüs. Als sich der Tooltip, als vermeintliches Untermenü öffnete, versuchte er dieses auszuwählen. Der Tooltip mit dem keine Funktion verknüpft ist verschwand und nichts passierte.

Dieses Beispiel zeigt, wie die Erwartung des Benutzers, welche auf den Erfahrungen mit der PC-Plattform basiert, zu einer fehlerhaften Interpretation der Anzeige führt. Erschwerend kommt hierbei hinzu, dass dies für das Menü *Neu* nicht konsistent innerhalb der Anwendung umgesetzt wurde, da an manchen Stellen ein Untermenü geöffnet wird und an anderen nicht.

Posteingang: Die Email Funktion von *Microsoft Outlook* versteckt sich in der deutschen Version unter dem Begriff *Posteingang*. Fast alle Teilnehmer hatten Probleme, unter diesem Namen auch die Funktion zum Erstellen neuer Emails zu finden. Interessanterweise zeigte sich hier ein deutlicher Einfluss der zuerst verwendeten Plattform bei dem Vorgehen zur Erstellung einer neuen Email Nachricht.

Die Teilnehmer der HHG hatten auf dem PocketPC nur wenige alternative Möglichkeiten und wählten so nach einigem Zögern den Posteingang, um dort über das Menü eine neue Email zu erzeugen. Der Abschluss der Aufgabe erfolgte hier deutlich schneller und unproblematischer als bei der zweiten Gruppe. Diese direkte Strategie wurde von dieser Gruppe auch erfolgreich auf dem PC angewandt, wo dann ohne zu zögern der Weg über den Posteingang gewählt wurde.

Bei der PCG boten sich in der PC Anwendung viel mehr alternative Wege eine neue Mail zu erstellen (Menü:Neu, Symbolleiste:Neu, Kontextmenü, etc.). Vier von fünf Teilnehmern versuchten, die Email direkt aus dem Adressbuch zu erstellen. Ein Teilnehmer versuchte, das Briefsymbol (s. Abbildung 2.8 und 2.11) in der Adressmaske anzuklicken, was nichts bewirkte da es sich hierbei um ein passives Bild handelt.

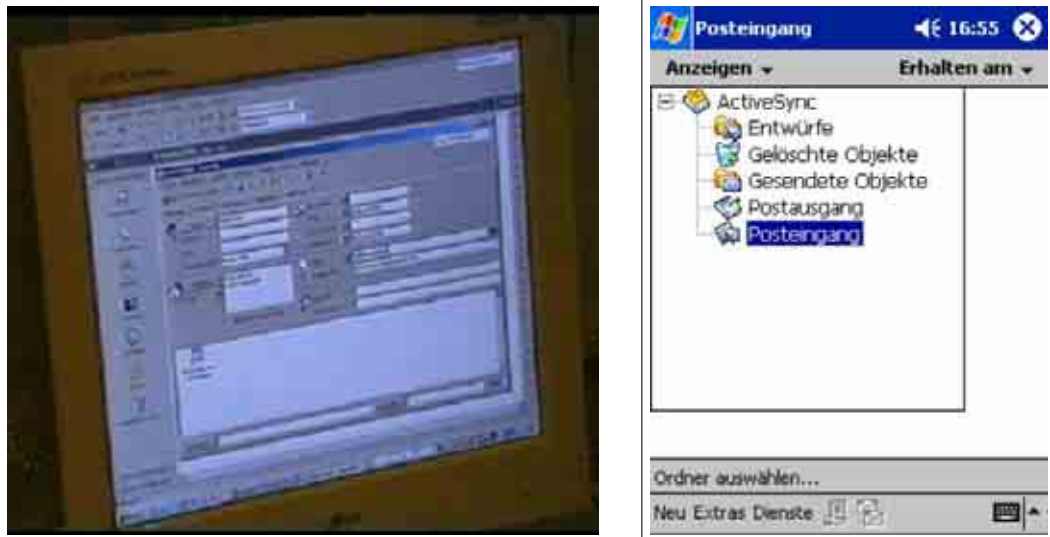


Abbildung 2.11: Ein Teilnehmer versucht auf dem PC eine neue Email aus dem Kontakt-Dialog zu erstellen indem er das Briefsymbol anklickt. Auf der rechten Seite wird Ansicht des Posteingangs auf dem Pocket PC dargestellt.

Diese Vielzahl an Strategien wurde von den Teilnehmern dann auch versucht, auf dem PocketPC anzuwenden, der einige dieser Alternativen nicht bot. Vor allem die Strategie eine Mail über das Adressbuch zu erzeugen, ist auf dem PocketPC deutlich schwerer zu realisieren, möchte man die Email an mehr als einen Adressaten versenden. Dies führte zu sehr langen Bearbeitungszeiten und zu vielen Fehlern bei zwei der Teilnehmer.

Dieses Beispiel zeigt, wie eine Übertragung der Strategien auf der Zielebene abhängig von der zuerst verwendeten Plattform mehr oder weniger erfolgreich sein kann. Das funktional eingeschränkte Gerät zeigt deutlicher den möglichen Interaktionsweg auf und ermöglicht es somit, die erfolgreiche Strategie auf das komplexere Gerät zu übertragen. Umgekehrt ist dies nicht der Fall. Dies ist insofern von großer Bedeutung, als dass der Wechsel vom komplexeren zum einfacheren Gerät wohl dem üblicheren Weg entspricht und somit potenzielle Bedienungsprobleme vorprogrammiert sind.

Die Kalender-Metapher: Bei der Bedienung der Kalenderanwendung auf dem PC zeigten sich zwei grundsätzlich verschiedene Herangehensweisen der Teilnehmer bzgl. der Erstellung eines neuen Termins. Während die eine Gruppe zur Erstellung des Termins direkt den *Neu*-Befehl oder Doppelklick verwendete und damit in die Eingabemaske (s. Abbildung 2.13) gelangten, wo ihnen alle erforderlichen Einstellungen zu Verfügung standen, unterschied sich das Vorgehen zweier Teilnehmer grundlegend. Diese beiden Teilnehmer gingen so vor, dass sie den Termin direkt in die Tagesansicht des Kalenders eintrugen (s. Abbildung 2.12). Dies ist auf der PC-Plattform einfach möglich, indem man die Uhrzeit markiert und die Eingabe über die Tastatur beginnt. Das Programm legt dann einen Termin mit Standardeinstellungen und mit dem eingegebenen Text als Betreff an. Um allerdings die weiteren Einstellungen, die in der Aufgabenstellung gefordert waren, einzustellen, wie z.B. Erinnerungszeit, Vertraulichkeit, etc., muss man die Eingabemaske öffnen.

Die Beobachtungsdaten zeigen, dass die zwei Teilnehmer, welche die direkte Eingabe wählten, offensichtlich keine Vorstellung von diesem Konzept hatten. Beide suchten nach diesen Einstellungsmöglichkeiten in der Kalenderoberfläche. Hierzu scrollten sie in der Tagesansicht nach oben oder unten in der Erwartung diese Einstellmöglichkeiten zu finden (*Einstellung von Erinnerung sollte als Icon vorhanden sein*). Erst ein Hinweis des Versuchsleiters führte dazu, dass die Eingabemaske gefunden wurde und die Aufgabe gelöst werden konnte. Ein Teilnehmer schloss daraus die Regel: *Aha, generelle Vorgehensweise ist also, erst den Termin zu erstellen und dann einzutragen*. Dieser Teilnehmer konnte dieses Wissen auf die zweite Plattform erfolgreich übertragen. Der andere Teilnehmer scheiterte an demselben Problem auch auf dem PocketPC.

Ein weiteres Problem, von dem sich vier der Teilnehmer irritiert zeigten, ist die Darstellung der Erinnerungszeit. Die Anwendung gibt kein graphisches Feedback über die eingestellte Erinnerung (lediglich eine Glocke, die anzeigt, dass erinnert wird). Viele der Teilnehmer konnten aufgrund der Anzeige nicht sagen, ob die Erinnerung nun *n* Minuten *vor* dem Beginn oder *für* *n* Minuten eingestellt war. Zwei Teilnehmer versuchten die Erinnerungszeit in der Tagesansicht direkt-manipulativ zu verändern.

Das bemerkenswerte an dieser Beobachtung ist die Möglichkeit, auf das zugrundeliegende mentale Modell zurück zu schließen. Es liegt nahe, das beobachtete Verhalten mit dem Vorgehen im klassischen Papierkalender zu vergleichen. Ähnlich wie bei dem Vorgehen der beiden Teilnehmer gibt es hier kein *verborgenes* Termin-Objekt, welches über eine Eingabemaske konfiguriert werden kann. Was zu einem Termin zu bemerken ist, wird hier direkt eingetragen. Das konzeptuelle Modell der Anwendungsentwickler differiert hier deutlich von dem mentalen Modell der beiden Anwender.

Bemerkenswert ist hierbei auch die Weise, auf welche die Entwickler der PocketPC-Anwendung diesem Problem begegnet sind: beginnt man hier einen Termin direkt einzutragen (was drei Teilnehmer so taten) öffnet sich direkt die Eingabemaske. Dies führte zu keinerlei Irritationen der Teilnehmer, leitete direkt zu den notwendigen Einstellmöglichkeiten und reduzierte somit mögliche Fehlerquellen.

Sukzessive Adaption: Ein weiteres Phänomen, welches bei der Analyse der Videodaten augenfällig wurde, war die Tendenz der Teilnehmer, ihre Eingabe schrittweise an die Plattform anzupassen: die sukzessive Adaption. Dies war insbesondere für die bezüglich der Eingabemöglichkeiten eingeschränkte PocketPC-Plattform der Fall. Die Benutzer zeigten meist die Tendenz eine Funktion über eine präferierte Eingabemöglichkeit aufzurufen. Die Flexibilität der Benutzereingabe, ein Gestaltungsmerkmal der Individualisierbarkeit direkt-manipulativer Oberflächen [Wes02], erlaubt es hierbei, dass verschiedene Benutzer sich verschiedene Strategien aneignen können.

Diese verschiedenen Strategien wurden auf der PocketPC zunächst versucht anzuwenden. Nur in wenigen Fällen war eine Übertragung ohne weiteres möglich, da z.T. die Eingabemöglichkeit Doppelklick nicht bestand oder eine Eingabe nicht zu dem erwarteten Ergebnis (z.B. Markieren durch Tippen mit dem Stift) führte. Die anfänglich große Varianz wurde also durch die Erfahrung auf der Mobil-Plattform kanalisiert. Eine hierbei erfolgreiche Strategie war die Verwendung des Kontextmenü. Dieses erzielte in der Funktionalität die beste Übereinstimmung zwischen den Plattformen. Diese Methode etablierte sich bei vielen somit schnell zur Standardeingabe (s. hierzu auch Abschnitt 2.2.8.2). Wurde eine gewünschte Funktion auch im Kontextmenü nicht gefunden musste in der Oberfläche exploriert und die Menüs abgesucht werden. Dies schien allerdings in der Regel vielmehr die letzte mögliche Lösung. Direkte Manipulation im Sinne von *Drag & Drop* wurde auf dem PocketPC nie angewendet.

Die sukzessive Adaption auf dem PocketPC bestand also aus einer Annäherung vom übertragenen Standardverhalten (z. B. Doppelklick) zu einem neuen Standardverhalten (z. B. Kontextmenü) und dem letztendlichen Notfallplan, der Suche in den Menüs und Symbolleisten der Oberfläche.

2.2.7.3 Zusammenfassung der Videoauswertung

Die Auswertung der Videodaten zeigte zunächst deutlich, dass Wechselwirkungen zwischen den verschiedenen Plattformen auftreten und dass die Gestaltung der Anwendung auf beiden Plattformen den Wechsel zwischen den Plattformen sowohl unterstützen, als auch behindern kann. Der Transfer zwischen den Plattformen findet auf allen Ebenen der Kommunikation zwischen Mensch und Computer statt. Somit ist auch jede dieser Ebenen anfällig für Übersetzungsfehler zwischen den Plattformen. Beispiele für Fehler auf allen Ebenen der Mensch-Maschine Kommunikation wurden beobachtet und beschrieben.

Die Art des Fehlers wird nach Reasons Fehlermodell bestimmt durch den Grad der bewussten Aufmerksamkeit und der Automatisierung der Ausführung. Die Zuordnung der beobachteten Fehler zu den verschiedenen Fehlerarten diente dazu, die Ursache des Fehlers zu spezifizieren und zu bestimmen, ob der Fehler aus der wissensbasierten Beurteilung der Situation, der Anwendung von übertragenen Regeln oder der Auslösung von Automatismen entstand. Vor allem die regelbasierten und

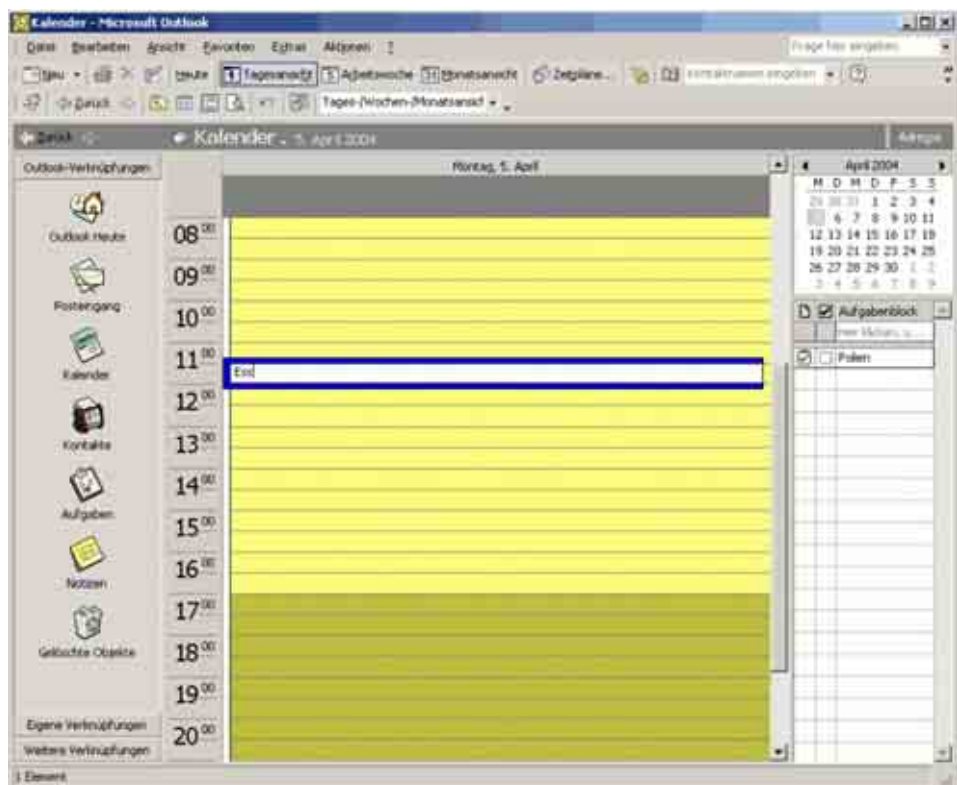


Abbildung 2.12: Eingabe eines neuen Termins direkt in die Tagesansicht des Kalenders.

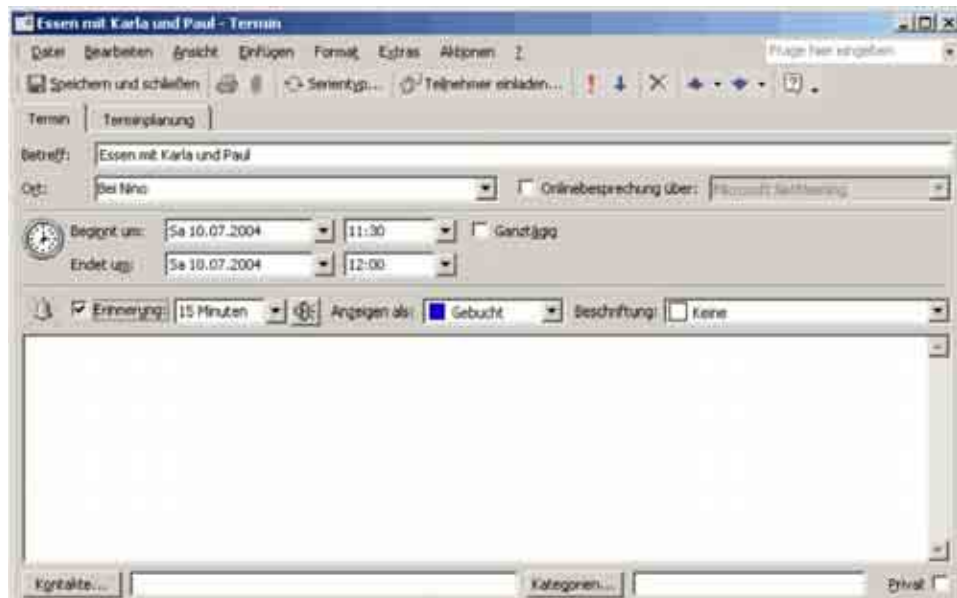


Abbildung 2.13: Eingabe eines neuen Termins über die Eingabemaske.

die automatisierten fertigkeitsbasierten Fehler spielen bei der Untersuchung der Wechselwirkung zwischen den Plattformen eine Rolle.

Die Untersuchung brachte folgende Kategorien von Unterschieden, welche den Wechsel zwischen den Plattformen erschweren, zu Tage:

Gestaltung von Eingabeelementen (Controls): Die inkonsistente Verwendung von ähnlichen Eingabeelementen mit verschiedener Funktionalität auf den beiden Plattformen führte, z.B. in dem Fall des Dropdown-Pfeils im Adressdialog (s. Abschnitt 2.2.7.2) zu Verwirrung und Fehlern. Die fehlerhafte Übertragung des Wissens zwischen den Plattformen scheint anhand der Videodaten eindeutig.

Benutzereingaben: Das Ergebnis einer Eingabe durch den Benutzer ist anhand des Vorwissens von der Benutzung der jeweils komplementären Plattform nicht eindeutig vorhersagbar. Setzt man den einfachen Mausklick mit dem Tippen des Stifts auf den Bildschirm gleich, verhält sich das Mobilgerät in vielen Fällen völlig unerwartet (s. hierzu auch Abschnitt 2.2.8.2). Der selbe Effekt tritt bei der Verwendung des virtuellen Keyboards auf (s. Abschnitt 2.2.7.2). Die sukzessive Adaption (s. Abschnitt 2.2.7.2) zeigt, dass auch effizientes und gut geübtes Standardverhalten aufgegeben werden musste.

Inkompatibilität von Design-Grundlagen: Die Design-Grundlagen der beiden Plattformen scheinen in manchen Fällen nicht nur unabhängig zu sein, sondern sich vielmehr zu widersprechen. Der Benutzer wird dadurch genötigt, widersprüchliche Regeln für die beiden Plattformen zu lernen. Ein Beispiel hierfür ist die Platzierung der OK-Schaltfläche an der selben Stelle wie das Schließen-Kreuz bei der PocketPC-Plattform (s. Abschnitt 2.2.7.2). Weitere Beispiele sind die Verwendung von Unterdialogen ohne Fensterrahmen (z.B. Adresseingabe), wie dies auf dem PC nie der Fall ist, oder die inkonsistente Verwendung von Menüs, welche wie Schaltflächen direkt Aktionen auslösen können (s. Abschnitt 2.2.7.2).

Auch hier finden sich somit die verschiedenen Ebenen der Mensch-Maschine-Kommunikation wieder, die physikalisch-lexikalische, die syntaktische und, betrachtet man die Abschnitte 2.2.7.2 und 2.2.7.2, auch die semantischen und darüber liegenden Ebenen. Plattformübergreifendes Design muss folglich alle diese Ebenen gleichermaßen berücksichtigen. Um die Durchgängigkeit zwischen Plattformen zu unterstützen, müssen alle diese Ebenen konsistent gestaltet werden.

2.2.8 Auswertung der Untersuchungen zum mentalen Modell

Die Methoden zur Erfassung des mentalen Modells des Benutzers können jeweils nur Teilaspekte beleuchten. So beschränkt sich die *direkte graphische Externalisierung* vor allem auf das konzeptuelle Gesamtbild (*wie hängt das ganze zusammen*), die *Teach Back Methode* vor allem auf das prozedurale Anwendungswissen (*wie kann ich*) und die *Antizipationsmethode* beleuchtet in besonderem Maße die syntaktische Ebene der Interaktion (*womit kann ich was*). Somit finden sich auch hier Normans Kommunikationsebenen wieder. Im folgenden werden die Ergebnisse dieser Untersuchungen im Kontext der Ergebnisse der Videodaten vorgestellt und diskutiert werden.

2.2.8.1 Direkte Graphische Externalisierung

Bei der direkten graphischen Externalisierung handelt es sich um ein Verfahren zur Erfassung der Struktur der internen Repräsentation welche der Benutzer sich von der Anwendung aufgebaut hat. Sie wird also im Anschluss an die Benutzung des System erhoben. Die Instruktion, die die Teilnehmer erhielten, war wie folgt:

Eindrücke lassen sich häufig schwer verbal ausdrücken. Aus diesem Grunde möchten wir Sie bitten, eine **grobe Skizze von Ihrem „inneren Bild“ der Anwendung** zu machen.
Bitte zeichnen Sie auf, wie die Anwendung in Ihrer Vorstellung aussieht...

Die Ergebnisse dieser Methode, welche in diesem Kontext erstmalig eingesetzt wurde, zeigen deutlich die Unterschiede in der Wahrnehmung der Anwendung auf beiden Plattformen. Zunächst einmal

vermitteln die handgezeichneten Skizzen einen Eindruck davon, in wie weit die Anwendung sich als strukturierter logischer Prozess darstellt und in wie weit der Benutzer diese Struktur zu erfassen vermochte.

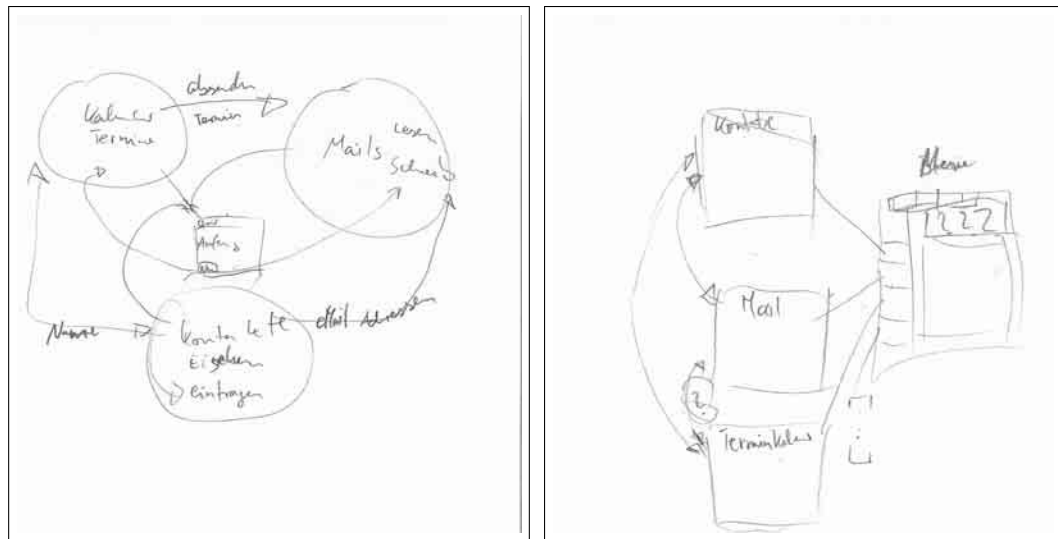


Abbildung 2.14: Anwendungsstruktur, wie sie sich einem Teilnehmer darstellte. Auf der linken Seite die Darstellung für das Mobilgerät, auf der rechten Seite die für den PC.

Abbildung 2.14 zeigt deutlich den Unterschied zwischen den beiden Anwendungen bzgl. der wahrgenommenen Struktur. Während der linke Teil der Abbildung, welcher die Anwendung auf dem Mobilgerät beschreibt, einen relativ ungeordneten Aufbau vermittelt, zeigt die rechte Darstellung der Anwendung auf dem PC eine recht klare, geordnete Struktur der Programmteile, welche einheitlich über eine zentrale Oberfläche gewählt werden können. Dieser Unterschied findet sich prinzipiell in fast allen Darstellungen wieder. Ebenso ist die Darstellung der drei Grundkomponenten *Kontakte*, *Termine* und *Email* konsistent über alle Darstellungen zu finden.

Die Weise, in welcher die Verknüpfung zwischen den Programmteilen dargestellt wird, scheint ebenfalls die Wahrnehmung der Benutzer sehr gut widerzuspiegeln. Während die Darstellung für den PocketPC ebenfalls sehr unstrukturiert scheint und verschiedene Verknüpfungslinien zwischen den Programmteilen beschreibt (*Namen können im Kalender verwendet werden*, *Email Adressen im Email Programm*, *ein Termin sendet Emails*,...), wird für den PC eine klare und uneingeschränkte Verknüpfung zwischen den Programmteilen angenommen.

Die Homogenität der Anwendung scheint für den PC deutlich besser wahrnehmbar zu sein. Wie Abbildung 2.15 zeigt, wurde die Anwendung auf dem Mobilgerät als eine unstrukturierte Ansammlung von Einzelfunktionen wahrgenommen und nicht als geordnetes Ganzes wie die PC-Plattform. Auch diese Darstellung fand sich in zwei Fällen wieder. Deutlich ist auch hier die wahrgenommene Nähe der Adressen- und der Email-Funktion. Das mentale Modell des Benutzers stellt diese beiden Komponenten in sehr engen Zusammenhang, eine Tatsache, welche sich auch in den Videodaten wiederfand. Für das Design einer solchen Anwendung gibt dieses wertvolle Hinweise, wie Funktionen stärker verwoben werden sollten. Weiterhin zeigt die symbolische Darstellung als *Topf in den alles hineingeworfen wurde*, dass diese Verbindung der Funktionen auf dem PocketPC vermisst wurde.

Abbildung 2.16 zeigt, wie sich die Wahrnehmung der Übersichtlichkeit der Anwendungen unterschied. Der Teilnehmer fühlte sich durch die Fülle von Unterfenstern und Dialogen, welche sich in der PC-Anwendung öffneten, beeinträchtigt. Die grundsätzlich klare Struktur der Oberfläche mit dem zentralen Übersichtsfenster und der Auswahlleiste wurde durch die diversen Unterdialoge zur Auswahl von Adressen, Eingabe von Adressen, etc., gestört. Hingegen wurde von diesem Teilnehmer die Übersichtlichkeit der PocketPC Anwendung durch die Eigenständigkeit der Komponenten positiv wahrgenommen. Aufgrund der mangelnden Fähigkeit, mehrere Fenster gleichzeitig anzuzei-

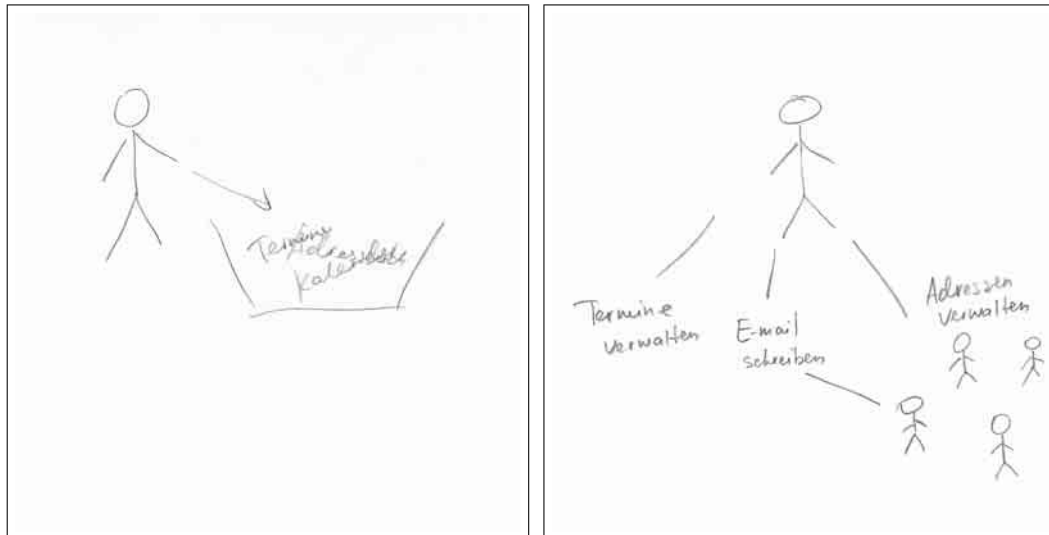


Abbildung 2.15: Homogenität der Anwendungen, wie sie von einem Teilnehmer wahrgenommen wurde. Auf der linken Seite die Darstellung für das Mobilgerät, auf der rechten Seite die für den PC.

gen, schließt sich die Verwendung von vielen Unterdialogen auf dem PocketPC freilich auch aus. Die Wahrnehmung der PocketPC Plattform als übersichtlicher, da weniger komplex, zeigt sich auch in den Darstellungen anderer Teilnehmer.

Zusammenfassend lässt sich feststellen, dass die Methode der direkten graphischen Externalisierung sehr gut geeignet ist, um auf einfache und schnelle Art den Gesamteindruck der Anwendung und der Anwendungsstruktur zu erfassen. Die Darstellung der Anwendungsteile gibt ein direktes Feedback hinsichtlich des Erfolges mit dem das konzeptuelle Modell vermittelt werden konnte. Die Homogenität sowie die Übersichtlichkeit der Anwendung schlägt sich direkt in der graphische Darstellung nieder.

Die Interpretation der Ergebnisse bleibt in einem gewissen Maße zwar spekulativ, jedoch zeigt die Übereinstimmung der Ergebnisse mit den Videodaten das Potential der Methode auf. Aufgrund der explorativen Natur der Methode erscheint eine detaillierte Validierung der Methode zunächst nicht nötig. Allerdings erscheint es lohnenswert, eine weitere Untersuchung der Methode in Folgestudien anzustreben.

Bezüglich des Vergleichs der Anwendungen auf den beiden Plattformen finden sich auch in den Ergebnissen der graphischen Externalisierung die bekannten Hauptunterschiede wieder. Die Gesamtstruktur der PocketPC Anwendung ist weniger transparent als die Struktur der PC Anwendung, hierdurch fällt es schwerer Verknüpfung zwischen den Anwendungen zu nutzen und vom Vorteil einer integrierten Anwendung zu profitieren. Dies schlägt sich auch in der Wahrnehmung der Homogenität der PocketPC Anwendung negativ nieder. Aufgrund der Unterschiede in der wahrgenommenen Struktur liegt die Vermutung nahe, dass eine gemeinsame Struktur beider Anwendungen und die Nutzung eines plattformübergreifenden Wissens schwer zu realisieren ist. In keine Richtung scheint sich die Struktur der Anwendungen zu decken.

2.2.8.2 Teach Back Methode

Bei der *Teach Back* Methode wird der Teilnehmer aufgefordert, das Gelernte einem imaginären Dritten (repräsentiert durch den Versuchsleiter) zu erklären. Anhand konkreter Fragestellungen (*wie kann ich...?*), muss der Teilnehmer artikulieren, wie er selbst eine Aufgabe gelöst hat und auf welche Probleme er dabei gestoßen ist. Über die Wahl des Vorgehens, die Gewichtung der verschiedenen Aspekte, die Wahl der Terminologie und die zu Tage tretenden Wissenslücken lassen sich konkrete Rückschlüsse auf das mentale Modell des Benutzer ziehen. Der Fokus liegt hierbei freilich auf auf

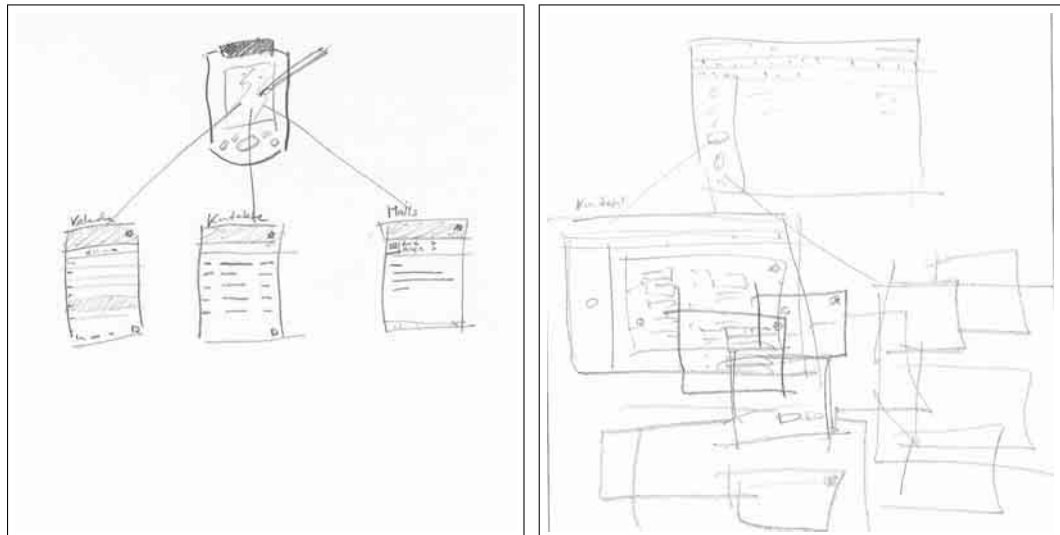


Abbildung 2.16: Übersichtlichkeit der Anwendungen, wie sie von einem Teilnehmer wahrgenommen wurde. Auf der linken Seite die Darstellung für das Mobilgerät, auf der rechten Seite die für den PC.

dem prozeduralen Wissen der syntaktischen, lexikalischen und physikalischen Ebenen der Mensch-Maschine-Kommunikation.

Zur Auswertung der *Teach Back* Methode wurden die handschriftlichen Skizzen gemeinsam mit den Videoaufnahmen, welche während der Erklärung gemacht worden waren, betrachtet. Besonderheiten, fehlerhafte Aussagen und Bemerkungen wurden getrennt für die vier Fragestellungen (*Wie schlage ich einen Termin nach?*, *Wie suche ich eine Telefonnummer heraus?*, *Wie trage ich einen neuen Termin ein?* und *Wie lösche ich einen Termin?*) notiert und auf Gemeinsamkeiten analysiert. Drei der vier Fragestellungen bezogen sich auf Aufgaben, die die Teilnehmer selbst durchgeführt hatten, eine Frage (Termin löschen) war neu und erforderte den Transfer vorhandenen Wissens.

Die Aussagen der Teilnehmer lassen verschiedene thematische Schwerpunkte identifizieren. Zum einen zeigte sich hier deutlicher als in der Videobeobachtung, dass den Teilnehmern die Konsequenz von Eingaben (Maus-/Stift-Klick, Doppelklick, gedrückt halten, rechte Maustaste, etc.) aufgrund der Inkonsistenzen zwischen den Plattformen häufig nicht klar war. Ein Teilnehmer stellte fest, dass es bei der PC-Anwendung möglich war, etwas zu markieren „ohne, dass man direkt gezwungen wird in die ‚Detail-View‘ zu gehen“. Andere Teilnehmer wurden unsicher, wenn es um die konkrete Eingabe ging um z. B. einen neuen Termin zu erzeugen. Einige meinten sich zu erinnern, dass ein einfacher Klick auf dem Mobilgerät genügen würde. Wieder andere vermuteten den, auf dem PocketPC überhaupt nicht verfügbaren, Doppelklick als die richtige Lösung.

Als recht einheitliche Vorgehensweise wurde die Verwendung des Kontextmenüs auf beiden Plattformen gewählt und weitergegeben. Das Kontextmenü, welches auf dem PC durch die rechte Maustaste und auf der PocketPC Plattform durch den *rechten Stift* (so ein Teilnehmer), d. h. durch gedrückt halten des Stiftes, aufgerufen wird, steht auf beiden Plattformen zur Verfügung. Diese Eingabe wurde von den Teilnehmer vornehmlich zur Erstellung neuer Einträge oder zum Löschen genutzt. Eine mögliche Erklärung könnte (neben der generellen Verbreitung auf dem PC) sein, dass sie auf beiden Plattformen zu ähnlichen Resultaten führt, wenn auch der Umfang auf dem PocketPC deutlich reduziert ist. Zum anderen kündigt das Kontextmenü das Ergebnis der Eingabe durch die Beschriftung der Menüoption an und macht es daher für die unbekannte Plattform vorhersagbar. Die Gefahr unliebsamer Überraschungen wird so geringer gehalten.

Die Validität der *Teach Back* Methode zeigt sich darin, dass sich die Fehler und Problemen auf den Videoaufnahmen in den Aussagen der Teilnehmer wiederfanden. Die fehlerhafte Wahrnehmung der Tagesansicht als Monatsansicht durch zwei Teilnehmer (s. Abschnitt 2.2.7.2) fand sich auch in deren Instruktion wieder. Ebenso wurde die Verwendung der *Return*-Taste zum Speichern von Terminen

auf dem Mobilgerät als normales Vorgehen bewertet. Auch das Vorgehen auf dem Mobilgerät über Menü / Neuer Kontakt in das Adressbuch zu gelangen, wurde weitergegeben, obwohl diesem Teilnehmer klar war, dass es sich nicht um die beste Lösung handelte. Die fehlerhaften Vorgehensweisen hatten sich also bereits zu festem Wissen manifestiert und würden weiterhin zu Fehlern führen.

Bezüglich der Terminologie fällt auf, dass das Konzept des *Startmenüs* nicht bei Allen korrekt vom PC auf den PocketPC übertragen wurde. Ein Teilnehmer setzte das Startmenü auf dem PocketPC mit dem Menüpunkt *Datei* auf dem PC gleich. Ein Anderer übertrug die Vorerfahrung vom Apple Macintosh Betriebssystem auf das Mobilgerät und bezeichnete das Menü oben links mit *Programme* wie dies dort üblich ist.

Bezüglich der Darstellung wurde erstaunlich häufig auf die Wahl einheitlicher oder verschiedener Icons auf beiden Plattformen verwiesen. Das einheitliche Symbol für den Kalender wurde positiv hervorgehoben, während beispielsweise die unterschiedliche Bezeichnung des Adressbuchs als *Rolodex* auf dem PC und Visitenkarte auf dem Pocket PC kritisiert wurde. Ein weiterer Hinweis betraf die Aufteilung des Kontakteingabe-DIALOGs auf dem Pocket PC. Dieser wurde als zu lange und unübersichtlich erachtet, was auch in Bezug auf die nachgeordnete Platzierung der privaten Daten kritisiert wurde. Eine Darstellung auf Karteikarten wurde alternativ vorgeschlagen. Diese Aussage deckt sich weitestgehend mit den Videobeobachtungen.

Zusammenfassend lässt sich feststellen, dass die *Teach Back* Methode hervorragend geeignet scheint, um explizite Aussagen zu Aspekten der Benutzbarkeit und der plattformübergreifenden Konsistenz zu erhalten. Die Aussagen sind natürlich bereits vom Benutzer reflektiert und können dadurch Verzerrungen und Interpretationen enthalten. Die Übereinstimmung mit den Videobeobachtungen belegt jedoch die Validität der Methode.

Die Teach Back Methode hat aufgezeigt, dass die Benutzer nicht über die Verwendung verschiedener Eingabemethoden im Klaren sind. Hier lässt sich anhand der Schilderungen der Teilnehmer ein deutlicheres Bild der Verunsicherung zeichnen, welche durch die offensichtliche Inkonsistenz des Verhaltens der Anwendung hervorgerufen wird. Ein Mausklick auf dem PC ist nicht gleich den Tippen eines Stiftes auf dem PocketPC. Der fehlende Doppelklick findet keinen Ersatz auf dem PocketPC. Als Ergebnis dieser Verunsicherung wird scheinbar in verstärktem Maße auf das Kontextmenü zurückgegriffen, welches die größte Übereinstimmung zwischen den Plattformen erreicht.

2.2.8.3 Antizipationsmethode

Bei der Antizipationsmethode wird erhoben, wie das syntaktische Wissen des Benutzers von der ersten Plattform auf die zweite übertragen wird. Hierzu wurden den Teilnehmern nach der Benutzung der ersten Plattform Abbildungen von Oberflächen der Anwendung auf der zweiten Plattform dargeboten. Es wurde abgefragt, was der Teilnehmer erwarten würde, wenn er an einer bestimmten Stelle klicken würde, bzw. wie er versuchen würde, eine bestimmte Aufgabe zu erfüllen. Für beide Gruppen wurden elf Abbildungen abgefragt. Diese Abbildungen korrespondierten jeweils für beide Gruppen. Im Folgenden werden die wichtigsten Ergebnisse dargestellt.

Die erste Frage betraf den Anfangsdialog. Hier sind alle Termine in den nächsten Tagen angezeigt. Gefragt war, was passieren würde, wenn man neben einen Termin klickt. Auf dem PC passiert nichts, da dort nur der Text in Form eines Hyperlinks aktiv ist. Auf dem Pocket PC wird der Kalender an dem betreffenden Tag geöffnet da die gesamte Zeile wie in einer Liste aktiv ist.

Das unterschiedliche Systemverhalten sollte sich nun in den Antworten der Teilnehmer widerspiegeln. Tatsächlich erwarteten alle Teilnehmer der HHG (welchen die Beispiele auf dem PC gezeigt wurden, da sie ja bislang den Pocket PC verwendet hatten), dass sich entweder der Kalender oder der Termin öffnen sollten. Anders bei der PCG (Pocket PC Beispiele, da Wechsel von PC), wo immerhin zwei Teilnehmer feststellten, dass der Punkt ausserhalb des aktiven Bereiches liegt. Drei Teilnehmer nahmen trotz der Erfahrung auf dem PC an, dass Details zu dem Termin angezeigt werden würden. Das Wissen, welches auf der ersten Plattform erworben wurde, wurde also in fünf von acht Fällen transferiert und führte damit zu fehlerhaften Annahmen.

Die vierte Frage betraf die Möglichkeit, im Kalender einen Tag vor oder zurück zu gehen. Zwei der Teilnehmer interpretierten die *Zurück*-Schaltfläche in der PC Anwendung als die richtige Option. Diese erlaubt allerdings, wie bei einem Internet Browser, die Navigation zu zuvor besuchten

Ansichten. Zwei Teilnehmer der PCG hätten die *Pfeil*-Schaltflächen zur Navigation zwischen Wochen verwendet. Hierbei handelt es sich offensichtlich weniger um eine fehlerhafte Übertragung zwischen den Plattformen, als um eine missverständliche Beschriftung der Schaltflächen. Dies war vermutlich auch bei der fünften Frage der Fall war, wo das Icon für die Besitzerinformationen im Startbildschirm des PocketPC (eine Visitenkarte) als Zugang zum Adressbuch interpretiert wurde.

Von besonderem Interesse ist das Ergebnis der sechsten Frage. Hier wurde der *Dropdown*-Pfeil (s. auch 2.2.7.2) aus der Kontakteingabe gezeigt. Die Interpretation der Teilnehmer der HHG war in allen Fällen richtig. Die Darstellung auf dem PC scheint somit eindeutig zu sein und nicht von den Erfahrungen auf dem Pocket PC überlagert zu werden. Tatsächlich wurde die Eingabemaske auch von keinem der Teilnehmer der ersten Gruppe zur Adresseingabe verwendet. Die Teilnehmer der PCG antworteten, wie dies aufgrund der Videodaten auch zu erwarten war, in drei Fällen, dass hier der Adresstyp umzuschalten sei. Zwei Teilnehmer vermuteten weitere Optionen zur Adresse hinter dem Pfeil. Dieses Ergebnis unterstützt zum einen die Interpretation der Videodaten und liefert ein weiteres Beispiel für die fehlerhafte Übertragung von Wissen zwischen der Plattformen aufgrund der Inkonsistenzen des Designs.

Weitere Fragen betrafen die Verwendung der *OK* bzw. *Kreuz*-Schaltfläche im oberen rechten Eck bei Pocket PC bzw. PC. Es zeigte sich auch hier, dass eine Fehlinterpretation der *OK*-Schaltfläche in vielen Fällen wahrscheinlich ist. In dem gegebenen Beispiel eines Dialoges zur Erstellung einer neuen Mail wurde in drei Fällen auf dem Pocket PC vermutet, *OK* könne die Mail absenden. Auf dem PC hingegen war das Kreuz als *Schließen*-Funktion bereits bekannt (s. Abbildung 2.11). Ein weiteres Beispiel für ein missverständliches Design auf dem Pocket PC fand sich im Email-Dialog, wo die Funktion zur Erstellung von Anhängen hinter dem Doppelpfeil für den erweiterten Email-Header vermutet wurde.

Zusammenfassend kann gesagt werden, dass die Ergebnisse der Antizipationsmethode die Ergebnisse der Videoaufzeichnung weitgehend stützten. Interessanterweise waren einige der Punkte, welche den Teilnehmern in der Untersuchung Probleme bereiteten, bereits in den Abbildungen enthalten. Einblicke, weshalb Benutzer manche Annahmen trafen, ließen sich mit Hilfe dieser Methode nur bedingt gewinnen. Die Teilnehmer der PCG schienen jedoch tatsächlich vor allem von den Erfahrungen auf dem PC abzuleiten, während die Teilnehmer der HHG vielmehr Grundwissen von der PC-Plattform einbrachten. Interferenzen mit der Erfahrungen auf dem Pocket PC waren nur eingeschränkt zu beobachten. Zur Identifizierung von missverständlichem Design und möglichen Problemen der Benutzbarkeit erscheint diese Methode sehr geeignet.

2.2.9 Diskussion

Die Analyse zeigte, dass sich eine große Anzahl der Probleme und Fehler mittels des Klassifikationsschemas von Reason [Rea90] einordnen ließen. Die Klassifikation von Fehlern als wissens-, regel-, oder fertigkeitsbasiert ist von großem heuristischem Nutzen, will man den Transfer von Wissen zwischen verschiedenen Endgeräten untersuchen. Selbst wenn in manchen Fällen keine klare Einordnung eines Fehlers in eine der Kategorien möglich ist, erlaubt dies dennoch, Rückschlüsse auf deren möglichen Ursprung zu ziehen. Dies ist von besonderer Bedeutung bei der Analyse des mentalen Modells des Benutzers und der Einflüsse auf dessen Bildung und Veränderung. Bei der Untersuchung der Ursachen von Fehlinterpretationen ist der Einblick in die Sicht des Benutzers von großer Wichtigkeit.

In der vorliegenden Studie führten einige anwendungsspezifische Gestaltungsaspekte dazu, dass Benutzer mentale Repräsentationen der Anwendung bildeten, die nicht zwischen Geräte transferiert werden konnten. Benutzer erreichten insbesondere dann nicht, geeignete Anpassungsregeln zu entwickeln, wenn keine konsistente Transformation der Anwendung zwischen den Geräte erkennbar war. Dies war der Fall bei der allgemeinen Struktur (z. B. Navigation zwischen den Teilen der Anwendung), bei den Dialogen (z. B. die Eingabemasken für Adressen) und bei den Inhalten (z. B. die inkonsistente Verwendung von Icons). In all diesen Fällen zeigt die Analyse einen klaren Zusammenhang zwischen Inkonsistenzen und Fehlern in der Bedienung. Interessanterweise konnten in einigen Fällen Fehler in beiden Richtungen verfolgt werden. Diese Fehler traten auf wenn Benutzer von PC auf PDA wechselte und umgekehrt.

Jedoch nicht nur das anwendungsspezifische Design beider Plattformen zeigte Inkonsistenzen auf, welche zu Fehlern führten. Auch das Design der Plattform an sich stellt in manchen Aspekten (z.B. Ort des Menüs, aktive Menüoptionen, Dialoge) Probleme für eine plattformübergreifende Bedienung und Entwicklung dar.

Wie diese Studie zeigt, kann plattformübergreifendes Design verbessert werden, wenn der Benutzer bei der Bildung und Festigung eines konsistenten mentalen Modells unterstützt wird. Er profitiert insbesondere dann, wenn er dieses Modell auf den verschiedenen Plattformen durch geringe Anpassungen weiterverwenden kann. Die folgende Liste stellt eine Zusammenfassung einiger Gestaltungsregeln dar, mittels derer eine plattformübergreifende Entwicklung verbessert werden kann.

- Das Gesamtbild einer Anwendung, welches beispielsweise durch die Navigation zwischen Teilen der Anwendung geformt wird, dient als Rahmen und Orientierungshilfe für den Benutzer. Wie die Methode der graphischen Externalisierung zeigte, traten hier große Unterschiede zwischen Plattformen auf. Entwickler plattformübergreifender Anwendungen sollten versuchen, das Gesamtbild über die Plattformen konstant zu halten.
- Die Verwendung von speziellen, nicht standardmäßig auf allen Plattformen vorhandenen Oberflächenelementen sollte vermieden werden, da hierfür spezielle Instanzen für die verschiedenen Geräte entwickelt werden müssen. In manchen Fällen kann es zu Konflikten mit gerätespezifischen Standards kommen (siehe Eingabemasken).
- Der Inhalt der Benutzerschnittstelle sollte konsistent über verschiedene Geräte erhalten bleiben [WBBG90, MS97]. Ikonische Information sollte stets dieselben Funktionen bezeichnen. Aus diesem Grunde sollten Icons so gestaltet werden, dass sie auf allen Geräten gezeigt werden können. Textuelle Information sollte während der Designphase bereits in verschiedenen Ausführungen entworfen werden.
- Gerätespezifische und geräteübergreifende Gestaltungsmerkmale müssen gegeneinander abgewägt werden. Im Falle der Vernachlässigung von Gerätestandards leidet die Konvergenz zwischen Anwendungen einer Plattform, bei deren übermäßiger Betonung kann die Konvergenz zwischen den Plattformen verloren gehen. Wo möglich, sollte der Entwickler deshalb auf systemspezifische Gestaltungsmerkmale verzichten. Aktive Menüeinträge auf oberster Ebene auf dem PocketPC könnten so z. B. durch einen Menüeintrag oder in eine Schaltfläche in der Symbolleiste ersetzt werden.
- Für den Fall, dass die Plattform keine eindeutigen Vorgaben über die Gestaltung eines Elements macht (wie dies z. B. bei den Dialogen der Fall ist, für die in *Pocket Outlook* drei verschiedene Darstellungsmöglichkeiten existieren), sollte der Entwickler sich auf eine Alternative festlegen und diese in konsistenter Weise über die gesamte Anwendung hinweg verwenden.

Ein weiteres Ergebnis der Untersuchung stellen die Beobachtungen zu den verwendeten Eingabemethoden dar. Diese scheinen zu einen sehr stark von der im Alltag verwendeten Plattform abzuhängen. In der Studie unterschieden sich hierbei insbesondere die Nutzer von Betriebssystemen wie Apple Macintosh von Microsoft Windows Nutzern. Gilroy und Harrison [GH04] stellen fest:

“Different styles of interaction often suit different devices most effectively. While the appearance of ubiquitous devices has brought forth a proliferation of innovative interactive techniques, the broad categories and aspects of style as, for example, identified by Newman and Lamming [NLL95] can still be applied. While a *keymodal interface* may be appropriate for a mobile telephone, with its limited screen and restricted keypad, a *direct manipulation (DM)* interface may be appropriate for a device based around touch / pen interactive techniques, such as current models of palmtop or tablet PCs.” [GH04]

Obwohl dies eine offensichtlich zutreffende Tatsache ist, muss aufgrund der Erkenntnisse aus der Studie auch festgestellt werden, dass nicht nur die passende sondern auch die *gewohnte* Interaktionstechnik angewandt wird. Die Anpassung der verfügbaren Interaktionstechnik an die Erfordernisse des Endgeräts ist also nur bedingt vorteilhaft. In der vorliegenden Studie, aber auch in einer zweiten Studie mit *Microsoft Pocket Word* (s. 7.2.2), konnte die Tendenz zur Nutzung des Kontextmenüs beobachtet werden. Dies kann dadurch erklärt werden da das Kontextmenü auf beiden Plattformen

verfügbar ist. Führt das Kontextmenü nicht zum Erfolg, wurden schrittweise, meist in einer festen Reihenfolge, verschiedene Alternativen ausprobiert bis dann der Zugang zu der gewünschten Funktion gefunden wurde.

Offensichtlich wurde der Funktionsumfang des Kontextmenüs auf dem PocketPC deutlich eingeschränkt, was der oben geschilderten Philosophie der Anpassung des Interaktionsstils entspricht. Kontextmenüs sind aufgrund der fehlenden rechten Maustaste auf dem PDA unpraktisch und können nur über eine Zeitverzögerung aufgerufen werden. Dennoch zeigte sich die Dominanz des Gewohnten. Dies stellt generell die Frage, welche Anpassungsstrategie zu bevorzugen ist, die Optimierung auf das Gerät oder die geräteübergreifende Konsistenz. Diese Frage, auch als Optimierungsproblematik bezeichnet, wird in den Abschnitten 5 und 6.2 noch ausführlicher diskutiert werden.

2.3 Anforderungen an die Benutzbarkeit

2.3.1 Benutzerwissen und Konsistenz

Wie die in Abschnitt 2.2 beschriebene Studie gezeigt hat, ist es wichtig den Benutzer beim Wechseln Plattformen zu unterstützen. Denis und Karsenty [DK04] nennen dies die Forderung nach der Wissenskontinuität oder *knowledge continuity*. Wissen welches ein Benutzer bei der Interaktion mit einer Applikation auf einem Gerät erlangt hat, sollte generell auch bei der Interaktion mit derselben Anwendung auf einem anderen Gerät nutzbar sein.

“One of the most important aspects of usability is consistency in user interfaces. Consistency should apply both within the individual application and across computer systems and even across product families.” [Nie89, S. 1]

Denis und Karsenty berufen sich hierbei auf die Ergebnisse von Holyoak *et al.* [HNM94], welche feststellten, dass Benutzer versuchen, gelernte Strategien mit Hilfe von Analogien auf neue Situationen anzuwenden (ähnliche Grundannahmen ziehen sich durch verschiedene theoretische Ansätze der kognitiven Psychologie von den Schematheoretikern und Konnektionisten [MR86] bis hin zu den regelbasierten Systemen und Produktionssystemen [And96, Wan93]). Im Falle der plattformübergreifenden Adaption bedeutet dies, dass Analogien zwischen den Geräten für den Benutzer konsistent wahrnehmbar sein müssen [Nie89], um ihn dabei zu unterstützen, erworbenes Wissen zur verfestigen. Mit anderen Worten bedeutet dies, dass die *Konsistenz der Anwendung* den Benutzer dabei unterstützt, sein mentales Anwendungsmodell über verschiedene Geräte hinweg aufrecht zu erhalten.

Gleichzeitig sollten allerdings auch die gerätespezifischen Designvorgaben befolgt werden, da diese die Konsistenz innerhalb eines Gerätes sicherstellen. Ansonsten beginnt der Benutzer mit jeder Anwendung von vorne. Konventionen für eine Plattform werden in der Regel in sog. Styleguides von den Systemherstellern festgelegt [App97, Mic03, Mic04].

Idealerweise stellt sich die Abbildung zwischen Geräten also als ein Kompromiss zwischen der Konsistenz innerhalb des Gerätes über die Anwendungen hinweg (*intra-device consistency*) und der Konsistenz innerhalb der Anwendung über die Geräte hinweg (*intra-application consistency*) dar. Die Konsistenz innerhalb des Gerätes hilft hierbei insbesondere, eindeutige und verlässliche Abbildungsregeln, also Strategien, wie die gelernten Analogien gesetzmäßig angepasst werden können, zu entwickeln.

2.3.2 Benutzbarkeit

Eine grundlegende Anforderung an technische Systeme ist dessen Benutzbarkeit (engl. *usability*). Obgleich des Begriff selbst für jeden verständlich und mit einer Bedeutung verbunden ist, existiert keine einheitliche Definition dieses Begriffs. Eason [Eas84] definiert die Benutzbarkeit wertneutral als die „Abhängigkeit [der] Benutzerreaktionen von den Eigenschaften des Computersystems, von Merkmalen der Benutzer und von den Aufgaben- und Tätigkeitsmerkmalen der Benutzer“ (zitiert nach [Wan93, S. 5]).

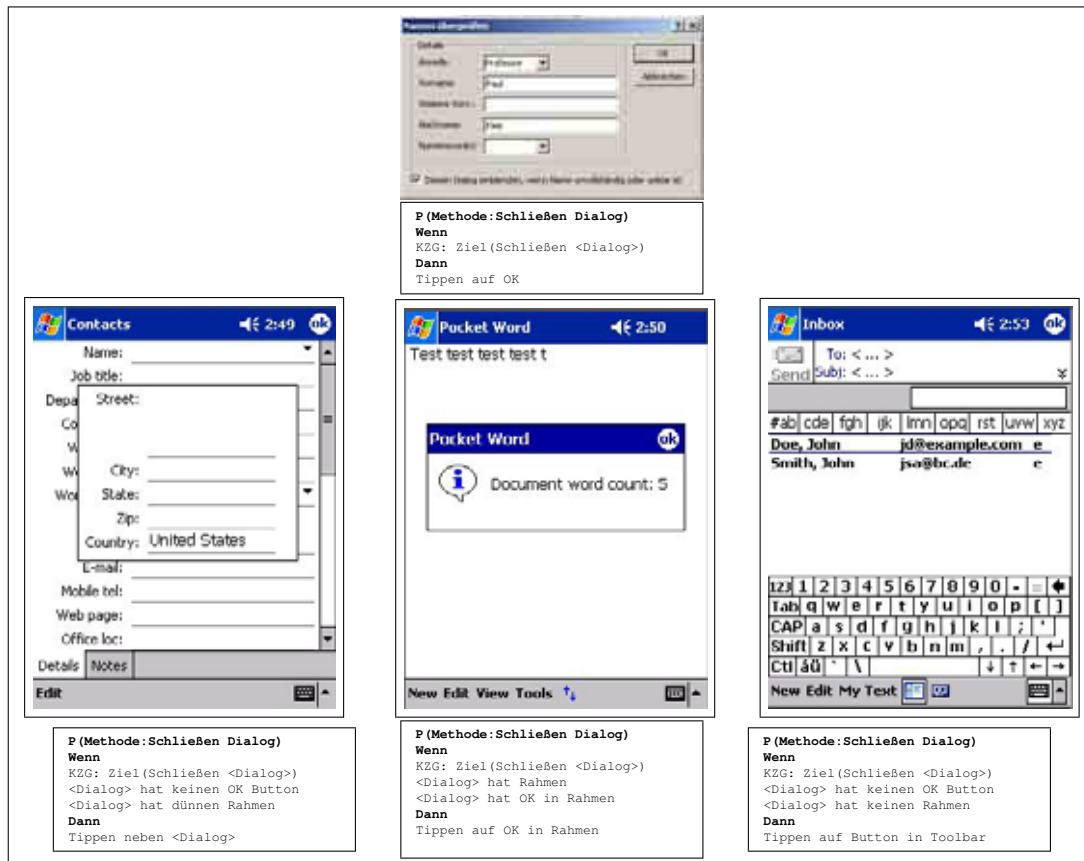


Abbildung 2.17: Beispielhafte Darstellung der Produktionen für die verschiedenen Typen von Dialogen und die Beziehung zur Konsistenz innerhalb Anwendung bzw. Gerät.

Shackel [Sha91] hingegen definiert Benutzbarkeit als „*the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfill the specified range of tasks within the specified range of environmental scenarios*“ und legt somit folgende Kriterien fest: Effektivität, Erlernbarkeit, Flexibilität und Einstellung. Shackel sieht hierbei die Benutzbarkeit, Nützlichkeit und Gefallen als unterschiedliche Dimensionen. Andere Ansätze fassen diese verschiedenen Aspekte hingegen zusammen.

Nielsen [Nie03] stellt den qualitativen Charakter der Benutzbarkeit in den Vordergrund, indem er sagt: „*Usability is a quality attribute that assesses how easy user interfaces are to use. The word 'usability' also refers to methods for improving ease-of-use during the design process*“ und gibt eine Liste von Kriterien der Benutzbarkeit vor [Nie93, Nie03] (siehe auch Shneiderman [Shn98, S.74f.] für seine *Acht Goldenen Regeln*).

Learnability (Erlernbarkeit): Wie leicht ist es für den Benutzer, eine Aufgabe beim ersten Mal richtig zu erfüllen?

Efficiency (Effizienz): Wie schnell kann ein erfahrener Benutzer Aufgaben ausführen?

Memorability (Einprägsamkeit): Wie gut lässt sich die Erfahrung und das Wissen über eine Anwendung über eine Zeit der Nichtbenutzung aufrechterhalten?

Errors (Fehlertoleranz und -anfälligkeit): Wie viele Fehler macht ein Benutzer, wie folgenschwer sind diese und wie leicht können die Folgen beseitigt werden?

Satisfaction (Zufriedenheit): Wie angenehm und zufriedenstellend ist die Benutzung?

Eine Reihe von internationalen Standards [ISO99a, ISO99b, ISO99c, ISO01, ISO99d] definieren die verschiedenen Aspekte der Benutzbarkeit und geben Regeln zu Erfassung und Umsetzung dieser Vorgaben an. Nach [AKSA03] lassen sich diese Standards in zwei Kategorien einteilen:

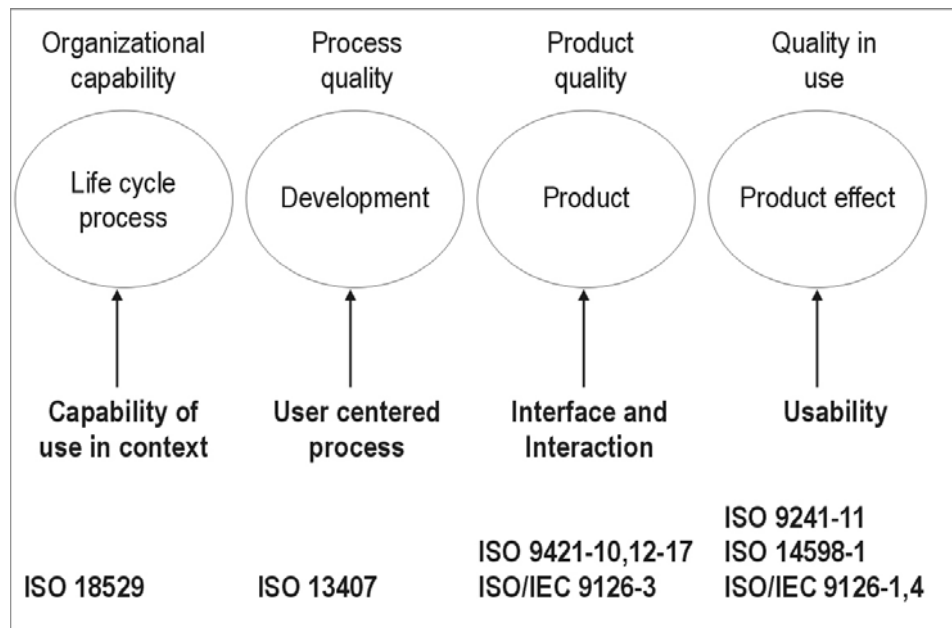


Abbildung 2.18: Kategorien der ISO Standards nach [AKSA03].

1. Produktorientierte Standards (ISO/IEC 9126, ISO 14598)
2. Prozessorientierte Standards (ISO 9241, ISO 13407)

Eine feinere Unterteilung bezüglich der adressierten Bereiche zeigt Abbildung 2.18.

Die *ISO/IEC 9126* Serie von Standards (2001-2003) besteht aus vier Teilen und behandelt die Aspekte der Benutzbarkeit von Informationssystemen als Produkte. Sie stellt eine der umfassendsten Klassifikationen zur Beschreibung der Softwarequalität dar und teilt diese in sechs Kategorien: Funktionalität, Verlässlichkeit, Benutzbarkeit, Effizienz, Wartbarkeit und Portabilität. Wie man unschwer erkennt, wurden hier einige der Kriterien Nielsens (s.o.) [Nie03] in eigene Kategorien gelegt. Die *ISO/IEC 9126-1*, 2000 definiert Benutzbarkeit wie folgt: „[*Usability is defined as...*] *The capability of the software product to be understood, learned, used and attractive to the user, when used under specific conditions*“. Kriterien für die Benutzbarkeit eines Softwaresystems sind somit dessen Verständlichkeit, Lernbarkeit, Bedienbarkeit, Attraktivität in einem definierten Kontext.

Die *ISO 9241* Serie von Standards fokussiert die Ergonomie im Arbeitsbereich der Büroarbeit und der damit verbundenen Prozesse. Diese Standards beschreiben die Anforderungen an Geräte, Software und Umgebungsbedingungen, die in einem Arbeitsumfeld herrschen sollten. In Teil 11 dieses Standards wird die Arbeit mit Computern beschrieben. Sie teilt sich auf in Materialanforderungen, Umgebungsanforderungen und Anforderungen an die Software. Letztere definiert sich wie folgt: „*software is usable when it allows the user to execute his task effectively, efficiently and with satisfaction in the specified context of use.*“ Die Kategorien in diesem Standard sind deutlich prozessorientiert und ordnen sich nach den Arten der Nutzung. Als Maße für Benutzbarkeit stehen hierbei die Effektivität, Effizienz und Zufriedenheit zur Verfügung. Abran *et al.* [AKSA03] argumentieren, dass eine Vereinheitlichung der Standards notwendig sei, um eine umfassende Definition der Benutzbarkeit zu erhalten und schlagen außerdem eine Erweiterung um Aspekte wie Sicherheit vor.

All diesen Definitionen ist eines gemeinsam: die Benutzbarkeit wird stets im Zusammenhang mit der Benutzung und den Eigenschaften des Benutzers gesehen. Somit ist klar, dass Benutzbarkeit in keinem Fall eine statische Eigenschaft eines Systems ist, sondern sich durch das Zusammenspiel mit dem Benutzer und Benutzungskontext definiert.

Van Welie *et al.* [Wel01, WVE99] verdeutlicht den Zusammenhang zwischen den Hauptkategorien der ISO Definition, deren Indikatoren in Form messbarer Größen und der Methoden, die zu deren Verbesserung herangezogen werden können (s. Abbildung 2.19). Als Basis dient jeweils das Wissen

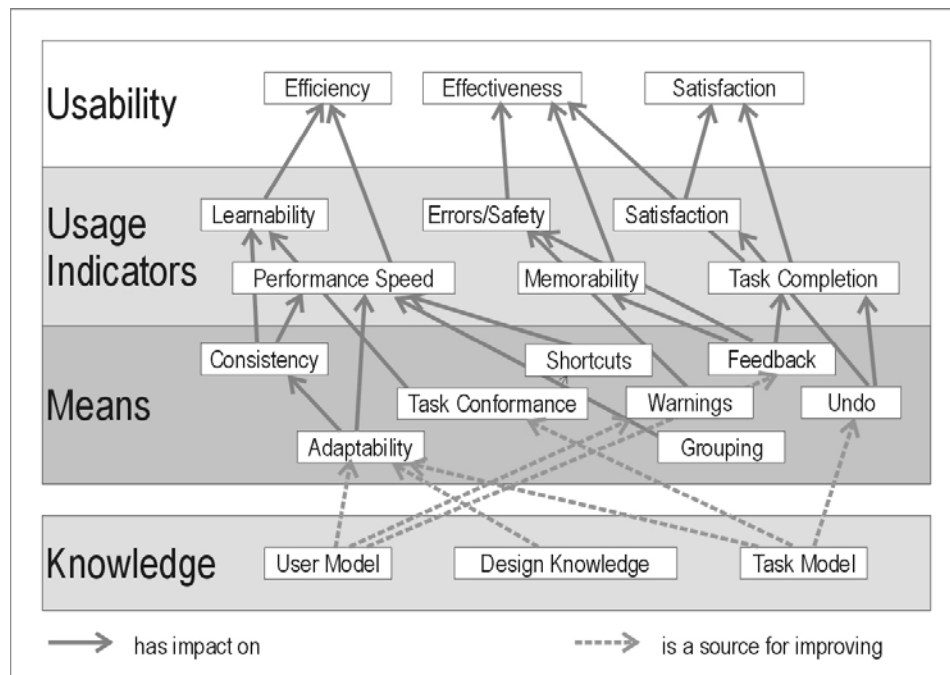


Abbildung 2.19: Ein Schichtenmodell der Benutzbarkeit nach [Wel01].

des Domänen-Experten, des Designers, des Usability-Experten oder des Software Entwicklers welches durch die verschiedenen Modelle repräsentiert wird. Die Kategorien der Benutzbarkeit (*Usability*) können durch die Indikatoren der Benutzung (*Usage Indicators*) nachgewiesen und gemessen werden. Diese wiederum lassen sich durch die Steuerung verschiedener Eigenschaften des Systems (*Means*) positiv beeinflussen. Das dazu nötige Wissen ist in Form verschiedener Modelle für das existierende Wissen (*Knowledge*) beeinflussbar. Das Modell von Welies ist, wie er selbst erwähnt, nicht komplett, dennoch eignet es sich, den Zusammenhang zwischen der Operationalisierung und Bereitstellung von Hintergrundwissen und der messbaren Verbesserung der Benutzbarkeit zu veranschaulichen. Van Welie *et al.* motivieren mit diesem Modell die Anwendung von Aufgabenmodellen bei der Entwicklung von Benutzerschnittstellen. Dennoch wird hier deutlich, dass die Integration von *höherwertigem* Wissen im Sinne von Designwissen, psychologischen Modellen und anderen, zu einer Verbesserung des Entwicklungsprozesses im Sinne der Benutzbarkeit führen kann.

2.3.3 Barrierefreiheit

In den vergangenen Jahrzehnten hat sich die Informationstechnologie von einem Anwendungsgebiet für Experten hin zu einem alltäglichen, für jeden präsenten und nutzbaren Bereich entwickelt, der unser Leben verändert hat. Wie jede neue Erfindung ist auch die Informationstechnik ein zweischneidiges Schwert: sie kann Menschen ebenso neue Möglichkeiten eröffnen wie sie sie mit Problemen konfrontieren kann, die diese zuvor nicht hatten. Eine Gruppe von Menschen musste dies sehr deutlich erfahren: Menschen mit Behinderungen.

Die technische Entwicklung orientiert sich am Durchschnitt der Bevölkerung. Jene, die nicht in dieses Schema passen werden häufig nicht berücksichtigt und müssen auf teure Sonderlösungen zurückgreifen oder lange auf die Verfügbarkeit warten.

Geräte mit Sprachsteuerung Menschen mit Sehstörungen im Internet zu surfen und neue Informationsräume zu erschließen. Gelähmte oder körperlich eingeschränkte Menschen können mit Hilfe von unterstützender Technik ihre Umgebung kontrollieren, sich selbständig bewegen und mit anderen Menschen kommunizieren und erhalten so neue Freiheiten und ein neues Gefühl der Lebensqualität. Jedoch machen sich diese Menschen von der Technik abhängig. Diese Technologien sind, obwohl sie existieren, nicht in allen Kontexten verfügbar. Eines der gravierendsten Beispiele stellt



Abbildung 2.20: Öffentliche Informationssysteme

das öffentliche Umfeld dar, in dem technische Installationen zunehmend die Rolle menschlicher Ansprechpartner übernehmen [Ric02a].

So ist der Behinderte im öffentlichen Umfeld, dort wo er Unterstützung am dringendsten benötigt, meist auf sich selbst gestellt. Sobald der Behinderte sein gut ausgestattetes Zuhause verlässt, steht ihm in den meisten Fällen nur noch die Technik zur Verfügung, die er mit sich führt und das ist meist wenig.

Sich durch eine unbekannte, sich ständig verändernde Umgebung zu bewegen, abgelenkt durch Lärm und wechselnde Lichtverhältnisse ist für viele Menschen mit Behinderungen eine Herausforderung. Informationstechnik ist ein Teil dieser unbekannten Umgebung und in manchen Fällen stellt sie die einzige Möglichkeit dar, an Dienste, Informationen oder Waren zu gelangen. Im öffentlichen Umfeld tritt Informationstechnik meist in Form von Informations- oder Verkaufsterminals auf. Diese Geräte sind in den meisten Fällen nicht nutzbar für Menschen mit Behinderungen. Man nehme als Beispiel einen Sehbehinderten, der vor einem Geldautomaten steht, welche über unbeschriftete Softkeys und keine Sprachausgabe oder Brailletastaturen verfügt (s. Abbildung 2.20). Eine Person in einem Rollstuhl sitzt vor einem Fahrkartenautomat und versucht die Knöpfe, den Geldschlitz, oder die Tastatur für die Eingabe der Geheimzahl zu erreichen um den nächsten Zug zu erreichen (evtl. mit behindertengerecht abgesenktem Boden).

All die hilfreiche Technik, die den Benutzer zu Hause oder auf der Arbeit unterstützt, steht ihm im öffentlichen Umfeld nicht zur Verfügung. Dies ist letztendlich die Kehrseite der Medaille der technischen Innovation. Anstatt zu integrieren verschärft Technologie in manchen Fällen die Trennung und die Benachteiligung von Menschen mit Behinderungen.

In den vergangenen Jahren haben immer mehr Regierungen und internationale Organisationen die Ausmaße des Problems erkannt und versuchen, mit Gesetzesinitiativen und Programmen zur Förderung der Barrierefreiheit [The03, Uni03, Wor94] die Integration voranzutreiben. Doch selbst wenn diese Gesetze und Vorschriften als Richtschnur und Absichtserklärung wichtig sind und den notwendigen gesetzlichen und gesellschaftlichen Rahmen geben, werden sie nicht in der Lage sein Lösungen zu produzieren. Im Falle der öffentlichen Terminal-Systeme stellt sich die Frage, wie sollten all die notwendigen individuell verschiedenen Interaktionsmittel zur Verfügung gestellt werden, damit jeder unabhängig interagieren kann? „*If a person cannot operate the world as it is currently designed [...]*“ [Van04, RH04b] muss entweder das Individuum oder die Welt sich ändern, oder es muss eine Brücke zwischen beiden gebaut werden.

Es wird nicht ausreichen, Hersteller von Terminal-Systemen zu verpflichten, ihre Geräte mit Sprachausgabe und Braillezeilen auszustatten, denn dies wird dem Menschen im Rollstuhl nicht helfen. Es hilft auch nicht, die Geräte so anzubringen, dass sie vom Rollstuhl aus einfach zu erreichen sind,

denn dann sind sie aus der Reichweite des blinden Nutzers und es wird ganz sicher schwer zu bedienen sein für den stehenden Nutzer ohne Behinderung, der sich dann bücken muss um das Gerät zu bedienen. Außerdem stehen häufig finanzielle und sicherheitsrelevante Faktoren [RRF04] (Vandalismus, Manipulation) einer Installation komplexer und teurer zusätzlicher Geräte entgegen. Kurz, es wird nicht möglich sein, einen Terminal zu entwickeln, der für die unendliche Vielfalt der möglichen Nutzer optimal zu bedienen ist.

Beschränkt man die Forderung nach Barrierefreiheit oder Zugänglichkeit (engl. *Accessibility*) greift man jedoch zu kurz. Ein zugängliches System ist nach der Definition von Vanderheiden [Van04] „*able to be used effectively by individuals either directly or with assistive technologies that they will have with them and can use when they encounter the environment, device, or system*“. Das Problem der Zugänglichkeit ist ein Thema, welches traditionell stets mit der Zugänglichkeit für besondere Nutzergruppen wie Älteren oder Menschen mit Behinderungen beschäftigt und deren besondere Anforderungen an Informationstechnologie betrachtet [Ste99]. Wie Stephanidis *et al.* [Ste01, SAS98] betonen, ist diese Zuordnung jedoch nicht weitblickend genug. Aufgrund der technischen Weiterentwicklung und aufgrund des Vordringens der Technik in unser alltägliches Leben, „*the range of the population which may gradually be confronted with accessibility problems extends beyond the population of disabled and elderly users*“ [SAS98]. Die Forderung nach Zugänglichkeit ist damit schlussendlich eine Weiterführung der Forderung nach Benutzbarkeit und schließt in der Folge immer stärker die soziale Anforderung an die Entwicklung von Informationstechnik ein.

Zugänglichkeit eines Systems erfordert, dass dieses in der Lage ist, sich an die Bedürfnisse des Benutzers anzupassen, an die Aufgabenstellung und den Kontext, wie auch an die technische Plattform die der Benutzer verwendet. Ein zugängliches System ist daher ein System, welches in der Lage ist, seine Benutzbarkeit bezüglich seines aktuellen Benutzers, der Aufgabe und der Systemkonfiguration zu optimieren. Die Benutzerschnittstelle als Mittel der Kommunikation zwischen System und Benutzer steht bei dieser Überlegung freilich im Vordergrund. Eine kontextsensitive Anpassbarkeit dieser Oberfläche bezüglich der einsetzbaren Interaktionsmittel, Ein- und Ausgabemoden und Informationen scheint somit eine zentrale Anforderung an ein zugängliches plattformübergreifendes System [Ovi03].

Norman stellt fest, dass „*Human action has two aspects, execution and evaluation. Execution involves doing something. Evaluation is the comparison of what happened in the world with what we want to happen*“ [Nor88]. Norman nennt die Distanz zwischen dem Zustand der realen Welt und der vom Benutzer wahrgenommenen Welt *Gulf of Evaluation* (Kluft der Beurteilung) und die Distanz zwischen der ursprünglichen Intention des Benutzers und dem tatsächlichen Effekt seiner Handlung *Gulf of Execution* (Kluft der Ausführung, s. Abbildung 2.21). Die Reduzierung dieser Entfernungen, d. h. die Überwindung der Kluft (*bridging the gulfs*) verbessert die Benutzbarkeit des Systems. Die Verbesserung der Benutzbarkeit für alle möglichen Benutzer führt zur universalen Benutzbarkeit (*universal accessibility*) des Systems. Folglich muss ein System, welches universal zugänglich sein will in der Lage sein, sich an die Bedürfnisse des Benutzers anzupassen um ihm bei der Überwindung der Kluft zu helfen.

Ein benutzbares System sollte versuchen beide Differenzen möglichst gering zu halten. Um dies zu erreichen kann das System an jeder der sieben Handlungsschritte ansetzen [Wan05], wie dies in Tabelle 2.3 beschrieben ist. Ein System kann sowohl Zielformulierung, Handlungsplanung, Sequenzierung als auch die Ausführung selbst unterstützen. Hierzu gibt es freilich verschiedene Methoden.

1. Motivation, Aktivierung und Zielsetzung
2. Wahrnehmung
3. Informationsintegration, Situationsbewusstsein
4. Entscheidungshilfen, Aktionsplanung
5. Handlungsausführung
6. Verarbeitung des Feedbacks.

Ebenso kann der Weg der Evaluation durch die Unterstützung der Wahrnehmung, der Interpretation und der Evaluation bzgl. des Ziel unterstützt werden. Assistenz kann hierbei gezielt einzelne Handlungsschritte unterstützen wie z. B. die Unterstützung der Interpretation durch Kontexthilfe, oder mehrere Phasen umklammern wie Automationshilfen oder Gedächtnishilfen.

1. **Ziel:** Der Benutzer muss entscheiden, welche Auswirkungen seine Handlung haben sollen.
 - *Gedächtnis:* Erinnerung an vorangegangene oder terminierte Ziele. Variiert von abwesend bis zu stark dominierend indem die möglichen Aktionen auf eine Liste der möglichen Ziele reduziert werden können (z.B. „Zuletzt verwendete Programme“)
2. **Ausführung:** Der Benutzer muss die Aktionen ausführen um sein Ziel zu erreichen.
 - a) *Intention:* Der Benutzer muss feststellen, ob das System ihm bei der Erreichung seiner Ziele helfen kann.
 - *Adaptive Unterstützung:* Das System kann aufgrund des aktuellen Handlungskontextes die Intention des Benutzers erschließen und dementsprechend unterstützen.
 - *Adaptive Hilfe:* Informationen zu möglichen Handlungen im aktuellen Kontext können in verschiedenen Detailtiefe und Ausführlichkeit gegeben werden.
 - *Modularisierung:* Eine komplexe Aufgabe kann vor dem Benutzer durch kleinere Teilschritte versteckt werden um so die Komplexität zu reduzieren.
 - b) *Handlungssequenzierung:* Der Benutzer muss die notwendigen Handlungen in eine sinnvolle Reihenfolge bringen.
 - *Leiten:* Der Benutzer kann durch den gesamten Prozess geleitet werden.
 - *Strukturierung der Eingabe:* Die Dialogstruktur kann durch Aufteilung und bessere Strukturierung an die Fähigkeiten und Anforderungen des Benutzers angepasst werden.
 - c) *Handlungsausführung:* Der Benutzer muss die physikalischen Handlungen einleiten um die Aufgabe auszuführen.
 - *Eingabemodalität:* Die vom Benutzer bevorzugte Eingabemodalität (kann z.B. von Kontext oder körperlichen Einschränkungen abhängen).
 - *Eingabegeräte:* Verfügbare Eingabegeräte (z.B. Bereitstellung eines virtuellen Keyboards), Konfiguration des Eingabegerätes (z.B. Verzögerung, Empfindlichkeit).
 - *Automatisierung:* Reduzierung des Eingabeaufwands z. B. durch Shortcuts, Makros, Fokus.
3. **Evaluation:** Vergleich des Handlungsergebnisses mit den ursprünglichen Zielen.
 - a) *Wahrnehmung:* Der Benutzer muss den aktuellen Zustand und Kontext wahrnehmen.
 - *Ausgabemodalität und Geräte:* Die bevorzugte Ausgabemodalität des Benutzers, bzw. die geeigneten Geräte.
 - *Ausgabesequenzierung:* Präsentation und Ausgabe kann an die Bedürfnisse des Benutzers und seinen Kontext angepasst werden, indem sie z.B. in kleine Schritte aufgeteilt oder reduziert wird.
 - b) *Interpretation der Wahrnehmung:* Der Benutzer muss den Zustand der Umgebung hinsichtlich seines Handlungszieles interpretieren.
 - *Status Feedback:* Rückmeldung über den aktuellen Zustand des Systems angepasst an den Benutzer und dessen Kontext (z.B. detaillierte Information für Experten, vs. Hinweise zur Behebung eines Fehlers bei Neulingen, akustische Tooltips für Blinde).
 - c) *Evaluation der Interpretation:* Der Benutzer muss die wahrgenommenen Effekte mit den ursprünglichen Zielen vergleichen und Konsequenzen ziehen.
 - *Ratgeber:* Alternative Wege zu dem ursprünglichen Ziel können für zukünftige Aktionen angegeben werden.
 - *Erinnerung:* Bei Abweichungen kann an das ursprüngliche Ziel erinnert werden und Hinweise zu dessen Erreichung gegeben werden.

Tabelle 2.3: Die unterschiedlichen Schritte des Handlungszyklus von Norman als Klassifikationschema für adaptierbare Systemeigenschaften, welche die Zugänglichkeit verbessern können. Ein technisches System kann den Benutzer in jedem Schritt der Handlung unterstützen. Diese Klassifikation weist den verschiedenen Handlungsphasen Systemfunktionen zu und beschreibt, wie diese an die Bedürfnisse des Benutzers angepasst werden können.

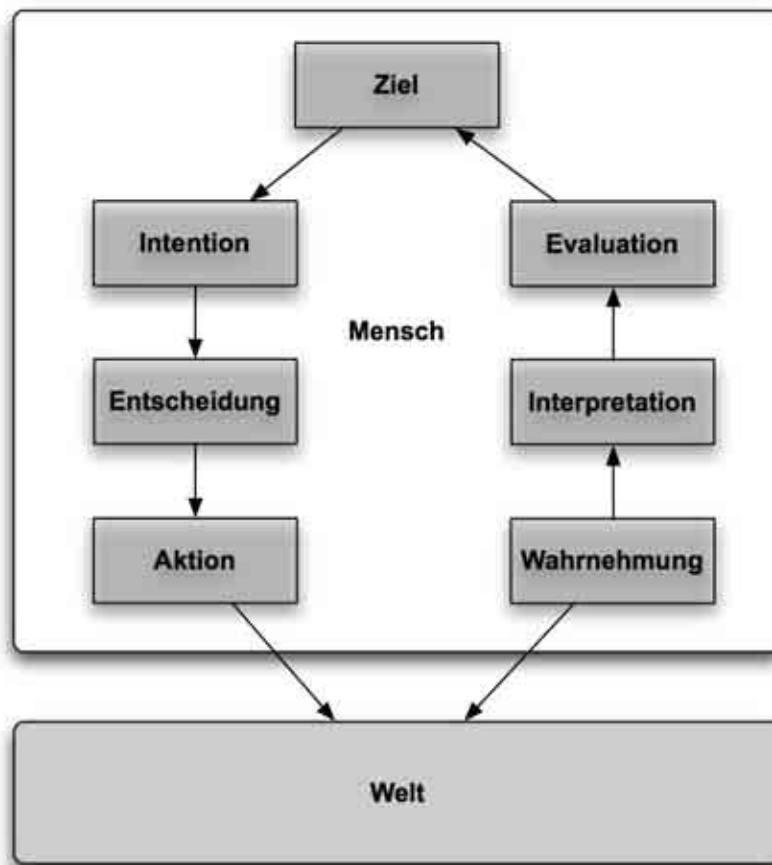


Abbildung 2.21: Das Evaluation-Execution Modell von Norman [Nor88].

Wandkes [Wan05] Klassifikationsschema bietet die Möglichkeit insbesondere die höheren Assistenzfunktionen auf Ebene der Handlungsplanung, Zielformulierung aber auch Evaluation weiter zu untergliedern in folgende Kategorien [WNPW02].

- *Angebotsassistentz*: informiert den Benutzer über alle zur Verfügung stehenden Optionen. Sie gewährleistet damit, dass der Benutzer überhaupt weiß, zwischen welchen Optionen er sich entscheiden kann.
- *Filterassistentz*: informiert den Benutzer über eine Auswahl der vorhandenen Optionen. Sie bietet dem Benutzer damit die Möglichkeit, das Angebot gezielt einzuschränken.
- *Beraterassistentz*: schlägt dem Benutzer eine in Frage kommende Option vor. Durch das System wird also nicht nur eine Vorauswahl getroffen, sondern diese wird soweit eingeschränkt, dass der Benutzer gar nicht mehr zwischen Optionen, sondern nur noch über das Annehmen oder Ablehnen dieser vorgeschlagenen Option entscheiden muss.

Tabelle 2.3 zeigt eine mögliche Anwendung der sieben Handlungsebenen Normans zur Klassifikation von Systemcharakteristiken der Unterstützung von Benutzern und Verbesserung der Zugänglichkeit [Wan05]. Methoden zur Verbesserung der Zugänglichkeit können entweder einzelne Systemeigenschaften adressieren (z.B. Anpassung der Eingabe, Kontexthilfe) oder mehrere Systemeigenschaften zugleich verbessern (z.B. Gedächtnis und Automatisierung).

Zwei Handlungsebenen erscheinen hierbei von besonderer Bedeutung: die Wahrnehmung (3a) und die Handlungsausführung (2c), da diese Ebenen die Systemgrenzen zwischen Benutzer und Welt im Handlungszyklus markieren. Ohne Unterstützung in diesen Ebenen werden insbesondere Menschen mit Behinderungen von der Benutzung solcher Systeme ausgeschlossen sein. Folgt man der Schlussfolgerung von Oviatt [Ovi03], welche herausstellt, dass „*multimodal interfaces have the potential*

to accommodate a broader range of users than the traditional interfaces” muss vor allem bei der Auswahl der Interaktionsmodalitäten die benutzerspezifische Adaption ermöglicht werden.

Zusammenfassend lässt sich feststellen: Zugänglichkeit ist also nicht mehr als eine unabhängige Eigenschaft eines Systems, sondern vielmehr als ein integraler Bestandteil desselben zu betrachten. Benutzer mit Behinderungen sollten nicht als eine besondere Benutzergruppe mit besonderen Anforderungen betrachtet werden, sondern vielmehr als ein Teil des Kontinuums menschlicher Vielfältigkeit [Shn98].

2.3.4 Zusammenfassung

In diesem Kapitel wurde anhand der Erkenntnisse aus einer explorativen Feldstudie (s. Abschnitt 2.2), der internationalen Standards und Normen zur Benutzbarkeit sowie einer Diskussion über die gesellschaftlichen Implikationen der Barrierefreiheit von Informationssysteme zentrale Anforderungen an die Benutzbarkeit eines plattformübergreifenden Systems erfasst. Die Forderung nach der universellen Zugänglichkeit (*Universal Accessibility*)[Ste01] von Informationssystemen, sowie die Erkenntnis, dass eine Kontinuität des mentalen Anwendungsmodells des Benutzers die Benutzbarkeit solcher Systeme verbessern kann sind in diesem Kontext als sich ergänzende Anforderungen von zentraler Bedeutung für die weitere Konzeption. Aus den oben ausgeführten Überlegungen lassen sich folgende Kern-Anforderungen an den Entwicklungsprozess zusammenfassen:

Wissenskontinuität: Der Benutzer sollte Wissen, welches er bei der Benutzung des Systems auf einer Plattform erworben hat auf einer anderen Plattform anwenden können.

Kontextadaptivität: Das System sollte in der Lage sein, sich auf die Bedürfnisse des Benutzers — in einer bestimmten Aufgabe auf einem bestimmten Gerät — anzupassen.

Multimodalität: Als Teil der Kontextadaptivität sollte die Darstellung der Benutzeroberfläche in verschiedenen Moden in der für den Benutzer günstigsten Konstellation möglich sein.

Benutzbarkeit: Die Benutzbarkeit des Systems sollte nach den Anforderungen internationaler Normen erfüllt und empirisch belegt sein.

2.4 Anforderungen an den Entwicklungsprozess

Die Entwicklung von Benutzerschnittstellen für verschiedene Plattformen soll in erster Linie zum Ziel haben, Entwicklungsaufwand, der bislang in die getrennte Entwicklung verschiedener plattformspezifischer Anwendungen geflossen ist, zu bündeln und Redundanzen zu reduzieren. Ziel muss es also sein, die Entwicklung anhand einer gemeinsamen, plattformunabhängigen Spezifikation zu ermöglichen. In Abschnitt 4 werden die wichtigsten existierenden Ansätze zur Spezifikation von Benutzerschnittstellen beschrieben. Diese unterscheiden sich nach Foley und Van Dam [FD82] in die vier Abstraktionsebenen der Windows-Systeme (s. Abschnitt 4.1), der sogenannten Toolkits, d. h. Sammlungen von wiederverwendbaren User Interface Elementen (s. Abschnitt 4.2) und der User Interface Management Systeme (UIMS), u. a. der modellbasierten Verfahren (s. Abschnitt 4.3).

Allen diesen Ansätzen gemeinsam ist das Vorgehen, User Interfaces basierend auf einer gemeinsamen abstrakten Spezifikation zu beschreiben und hieraus plattformspezifische Darstellungen automatisch zu erzeugen. Diese Ansätze unterscheiden sich in erster Linie durch die Komplexität der Spezifikation (z.B. Ebenen der Modellierung), durch den Grad der Abstraktion, durch die Syntax (Implementierung vs. textueller oder graphischer Spezifikation) und durch den Umfang bzw. Grad der Kontrolle bei endgültigen Generierung.

Im folgenden Abschnitt werden die wichtigsten Anforderungen an einen Entwicklungsprozess für plattformübergreifende Benutzerschnittstellen zusammengefasst (s. auch [Mye92]). Der Hauptaugenmerk liegt hierbei auf dem beschriebenen vorherrschenden Vorgehensmodell der automatischen Erzeugung von plattformspezifischen Darstellungen auf Basis einer abstrakten plattformübergreifenden Spezifikation. Novaks Regel [Nov04, Mol04] fasst diese Anforderungen treffend zusammen:

“Automatic Programming is defined as the synthesis of a program from a specification. If automatic programming is to be useful, the specification must be smaller and easier to write than the program would be if written in a conventional programming language.”
[Nov04]

Wie bereits oben dargestellt wurde, erfordert das benutzerzentrierte Vorgehen bei der Analyse der Anforderungen in dieser Arbeit die Fokussierung zum einen des Benutzers plattformübergreifender Systeme als auch zum anderen die Berücksichtigung des Entwicklers und Herstellers solcher Systeme [RKM88]. Neben den rein technischen Anforderungen (s. Abschnitt 2.5) spielen hier natürlich motivationale und Benutzbarkeitsaspekte, sowie die ökonomische Passung derartiger Prozesse in existierende Arbeits- und Vertriebsabläufe eine wichtige Rolle. Barnes und Huff [BH03] wenden die Kriterien des *Diffusion of Innovations* Modells von Rogers [Rog95, Rog97] an, um die Einführung des mobilen *iMode*-Systems zu analysieren und damit die notwendigen Bedingungen einer erfolgreichen Einführung von Innovationen zu identifizieren. Trætteberg [Tra04] überträgt dieses Vorgehen um Anforderungen an das modellbasiertem Design plattformübergreifender Benutzerschnittstellen zu formulieren. Diese Anforderungen sind in erster Linie organisatorischer Natur, die folgenden Abschnitte orientieren sich an den dort identifizierten Anforderungen, wie *Nutzen*, *Kompatibilität*, *Komplexität*, *Testbarkeit* und *Sichtbarkeit*.

2.4.1 Nutzen

Rogers [Rog95, Rog97] bezeichnet den relativen Nutzen (*relative advantage*) einer Innovation als einen wichtigen Faktor zu deren Adaption:

“Relative advantage is the degree to which an innovation is perceived as better than the idea it supersedes. The degree of relative advantage may be measured in economic terms, but social prestige, convenience, and satisfaction are also important factors. It does not matter so much if an innovation has a great deal of objective advantage. What does matter is whether an individual perceives the innovation as advantageous. The greater the perceived relative advantage of an innovation, the more rapid its rate of adoption will be.” [Rog97]

Relativer Nutzen setzt sich bezüglich des Entwicklungsprozesses aus mehreren Aspekten zusammen. Hierzu gehört zum einen sicherlich der finanzielle Nutzen des Herstellers und Entwicklers, welcher aufgrund optimierter Entwicklungsmethoden schneller und mit weniger Aufwand produzieren kann, d.h. die Rationalisierung der Entwicklung. Hoch strukturierte Entwicklungsmethoden führen meist zu einem Anstieg in Qualität und Fehlerfreiheit. Beispiele hierfür finden sich in der objektorientierten (OO) Entwicklung mit der Unified Modelling Language (UML, [BJR98]), der Verwendung von Entwurfsmustern und Algorithmen [GHJV94, Sed92] und automatischen Testmethoden wie Unit-Tests, usw. Man kann also sagen, dass als Nebeneffekt optimierter Entwicklungsmethodik die qualitative Verbesserung und die Reduzierung der Fehlerquote als relativer Nutzen abfällt.

Bei vielen Ansätzen zur Darstellung von plattformübergreifenden UI wird neben der Modellierung der Oberfläche selbst eine Anzahl von weiteren Modellen herangezogen, wie z.B. Benutzer-, Aufgaben-, Domänen- oder Umgebungsmodell. In einigen Fällen wird die Oberfläche sogar direkt aus diesen Modellen mit Hilfe von Regeln automatisch generiert ohne, dass diese explizit modelliert wurde. In anderen Fällen wird über einen oder mehrere Zwischenschritte eine Oberflächenmodellierung aus den zugrunde liegenden Modellen vorgenommen. Ziel ist hierbei ganz klar, die Oberflächen an verschiedenen determinierenden Faktoren anzupassen. Im Sinne der *Multiple User Interfaces* also eine optimale Anpassung an Benutzer, Situation und Ausgabe zu erreichen.

Das Problem hierbei ist in der Regel, dass durch die Erzeugung von zusätzlichen Modellen der Entwicklungsaufwand erhöht wird. Die Erzeugung von Modellen erfordert Spezialwissen, welches von Entwicklern und Designern nicht unbedingt zu erwarten ist und von der Aufgabe der Oberflächengestaltung weiter fort führt. Das zentrale Problem der multidimensionalen model-basierten Entwicklung ist die Tatsache, dass die zusätzliche Abstraktion durch Modelle zu einem deutlichen Mehraufwand führt. In manchen Fällen entsteht der Eindruck, dass so Anwendungen mehrfach aus verschiedenen Perspektiven programmiert werden müssen.

Für den Fall aktueller modellbasierter Methoden beurteilt Trætteberg [Tra04] den relativen Nutzen jedoch noch sehr skeptisch. Theoretisch führen diese Methoden zwar zu einer Reduzierung des Entwicklungsaufwandes, so Trætteberg, jedoch wird dies mehr als aufgehoben durch den Aufwand bei der Einarbeitung in die meist komplexen Methoden und Werkzeuge. Zusätzlich sind die meisten Werkzeuge, so denn überhaupt existent, meist reduziert im Funktionsumfang und im Anwendungsbereich. Spezielle Anpassungen gestalten sich meist als aufwändig bzw. unmöglich. Die Benutzbarkeit, bzw. Benutzerzentrierung dieser Systeme ist laut Trætteberg meist noch nicht besonders entwickelt. Ein scheinbar vernichtende Beurteilung des relativen Nutzens modellbasierter Methoden, welcher aufgrund des meist noch akademischen Ansatzes derartiger Anwendungsentwicklungsmethoden klar erklärbar ist, dennoch nicht davon ablenken sollte, dass viele hier angemahnte Anforderungen bei der weiteren Entwicklung Berücksichtigung finden sollten.

2.4.2 Kompatibilität

Kein Entwickler plattformübergreifender Systeme wird völlig unvorbelastet an eine neue Technologie herangehen. In der Beurteilung des Neuen steckt stets im Hintergrund die Erfahrung mit dem Alten. Die Weiterverwendbarkeit von existierendem Vorwissen, die Nutzung mühsam erarbeiteter Expertise, aber auch die Integration neuer Methoden hilft dabei die Schwelle einer Einführung neuer Methoden zu senken. Rogers [Rog95] bezeichnet dies als die Kompatibilität (engl. *compatibility*) einer Innovation und definiert diese wie folgt.

“Compatibility is the degree to which an innovation is perceived as being consistent with the existing values, past experiences, and needs of potential adopters. An idea that is incompatible with the values and norms of a social system will not be adopted as rapidly as an innovation that is compatible. The adoption of an incompatible innovation often requires the prior adoption of a new value system, which is a relatively slow process.”
[Rog97]

Die Vereinheitlichung der Benutzerschnittstellen für den Personal Computer hat vor allem durch den Erfolg der WIMP-Metapher eine schnelle und weitreichende Entwicklung genommen. Durch den Einsatz einheitlicher Interaktionselemente in den verschiedensten Anwendungen und die darauf folgende Kapselung graphischer Oberflächenelemente in sog. User Interface Toolkits und die modernen Oberflächenarchitekturen wie das *Model-View-Controller (MVC)* Modell [Sha90] wurde der Entwicklungsaufwand für graphische Oberflächen drastisch reduziert. So ging nach einer Studie von Wilson [Wil90] die Entwicklungszeit durch die Verwendung von Apples MacApp um ca. ein Fünftel zurück, und laut einer Studie von NeXT Computers reduzierte die Verwendung des NeXTStep Systems die Entwicklungszeit auf die Hälfte [Boo92]. Kompatibilität und herstellerübergreifend einheitliche Entwicklungsmethoden sind sicherlich ein wesentlicher Grund für den Erfolg der WIMP basierten Benutzerschnittstellen.

Entwicklungswerkzeuge, welche auf Toolkits aufsetzen, erlauben heute die direkt-manipulative Erzeugung von Oberflächen, ohne allzu große Programmierkenntnisse vorauszusetzen. In der Tradition von Sprachen wie Smalltalk ist die objektorientierte Erstellung von Oberflächen durch die Auswahl und Positionierung von Komponenten und deren Kombination zu komplexen Oberflächen kein Problem mehr. Die Trennung von Präsentation und Anwendungslogik, wie im MVC-Modell gefordert erlaubt darüber hinaus auch noch die einfache Erzeugung der Programmcode-Gerüste in welche die eigentliche Logik nur noch eingebunden werden muss.

Derartige Graphische *User Interface Builder* stehen heute für alle Plattformen zu Verfügung und im professionellen Kontext werden Entwicklungsumgebungen wie Microsoft Visual Studio mit dem Visual Designer [YJS03] oder das Open Source Projekt Eclipse [Hol04] flächendeckend eingesetzt. Zusammen mit weiteren Entwicklungshilfen bilden diese Umgebungen den modernen Arbeitsplatz für Anwendungsentwickler. Graphische Interface Builder ermöglichen auch die frühe Validierung und Korrektur von Oberflächen-Designs und somit schnelle Entwicklungsiterationen, da die Darstellung sofort sichtbar ist und auch von Mitarbeitern ohne Programmiererfahrung, d. h. Designer, Usability-Experten und Benutzer, beurteilt und sogar selbst angepasst werden. Graphische Interface Builder sind somit ein wichtiger Teil in interdisziplinären Entwicklergruppen, wie sie in heutigen Unternehmen häufig üblich sind.

Graphische und direkt-manipulative Entwicklungsumgebungen existieren auch für andere Modalitäten als den rein graphischen Oberflächen. Über die Visualisierung hierarchischer Bäume und andere Interaktionskonzepte werden auch z.B. für die Entwicklung von sprachbasierten Benutzerschnittstellen komfortable Möglichkeiten der Entwicklung geboten. Für die Entwicklung von plattformübergreifenden Benutzerschnittstellen allerdings sucht man derartige Werkzeuge bislang noch vergebens.

Myers [Mye95] klassifiziert die Entwicklungswerkzeuge zur Erzeugung von Benutzerschnittstellen, wie User Interface Management Systeme, Interface Builder und User Interface Entwicklungsumgebungen in vier Kategorien:

- Sprachbasierte Werkzeuge
- Anwendungsumgebungen (frameworks)
- Interaktive graphische Entwicklungswerkzeuge (auch *User Interface Builder*, s. o.)
- Modellbasierte Werkzeuge

Sprachbasierte Werkzeuge verwenden formale Spezifikationen wie State Transition Networks und Context-Free Grammars. Sie haben dabei den Vorteil, vor allem die Beschreibung von Interaktionen zu erleichtern, während die Graphischen Werkzeuge in der Regel bei der Spezifikation der Darstellung von Vorteil sind.

Anwendungsumgebungen (s. Abschnitt 4.2) bieten eine gute Balance zwischen detaillierter Kontrolle und Unterstützung für den Entwickler. Sie bieten bislang jedoch nicht den Grad von Abstraktion welcher notwendig ist um die Entwicklung für verschiedenen Plattformen zu unterstützen. In der Regel wird die Entwicklung von Oberflächen in Anwendungsumgebungen durch Werkzeuge erleichtert, welche direkte graphische Spezifikation erlauben (s.o.).

Interaktive graphische Entwicklungswerkzeuge können nach Myers, Hudson and Pausch [MHP00] in fünf Klassen unterteilt werden, deren individuelle Vor- und Nachteile ihren Einsatz in verschiedenen Kontexten erfordern.

- Prototypen-Werkzeuge
- Cards
- Interface Builders
- Datenvisualisierungswerkzeuge
- Graphische Editoren

Prototypen-Werkzeuge und graphische Editoren eignen sich zur direkten und unkomplizierten Spezifikation von Oberflächen oder Komponenten, welche ausschließlich der Vorentwicklung dienen und in Vortests genutzt werden oder dann als Ausgangsbasis für Anwendungscode dienen, existierende *Prototyping*-Ansätze zur Entwicklung von plattformübergreifenden Oberflächen werden im nächsten Abschnitt unter dem Begriff *Prototyping-Werkzeuge* zusammengefasst.

Modellbasierte Werkzeuge leiden vor allem an der indirekten Umsetzung von Eingaben. Die Veränderungen am Modell wirken sich nur mittelbar, abhängig von anderen Komponenten des Modells oder abhängig von den Darstellungsprozessen auf die tatsächliche Darstellung aus. Modellbasierte Ansätze benötigen daher in erster Linie unterstützende Werkzeuge zu Erstellung der Modelle. Die direkte Anpassung der Darstellung ist bei diesen Ansätzen nur schwer möglich. Den Alltag eines Entwicklers für plattformübergreifende Anwendungen beschreiben Ali, Pérez-Quñones und Abrams folgendermaßen:

“[...] when developers create an interface in an abstract language [...] that will be translated into one or more specific languages, they follow a process of trial and error.”
[FAPQA04]

Betrachtet man die vorherrschenden Vorgehensmodelle der Softwareentwicklung, zeigt sich ein Wechsel von den klassischen Software-Engineering Modellen wie dem *Wasserfallmodell* [Roy70], dem *Spiralmodell* [Boe86] oder dem *V-Modell* [DHM98], hin zu stärker benutzer- oder aufgabenorientierten Methoden des *Contextual* oder *Usability Engineering* [May99, BH98], oder den auf

Flexibilität und Geschwindigkeit optimierten *eXtreme Programming (XP)* [Bec99]. Andere Vorgehensmodelle, wie die des *Agile Usability Engineering* [GMR04] versuchen, beide Strömungen zu vereinen und die Vorteile beider Ansätze zu nutzen. Entsprechen die Methoden des modellbasierten Entwicklungsansatzes stärker der Vorgehensweise des Usability Engineering, so können Graphische Editoren und die toolkitbasierte Entwicklung besser auf die Anforderungen der flexiblen Entwicklung und der schrittweisen Verfeinerung, wie dies in den klassischen Vorgehensmodellen, bzw. dem XP üblich ist angepasst werden. Bei der Entwicklung plattformübergreifender Benutzerschnittstellen sollten daher beide Ansätze, im Sinne eines agilen Entwicklungsprozesses unterstützt werden.

Myers, Hudson und Pausch [MHP00] stellen fest, dass wesentlich für den Erfolg von *User Interface Toolkits* deren niedrige Schwelle, deren hohe Decke und deren Vorhersagbarkeit waren. Im Gegensatz hierzu zeichneten sich die meisten modellbasierten Ansätze durch ihre hohe Schwelle, die niedrige Decke und eine geringe Vorhersagbarkeit aus, was wesentlich zu deren geringer Akzeptanz beitrug. Mit Schwelle ist hierbei die Erlernbarkeit der Methode für den Entwickler, d. h. der Einarbeitungs- und Umsetzungsaufwand gemeint. Mit der Decke ist die Mächtigkeit des Ansatzes, d. h. wie umfangreich dessen Anwendungsbereich ist. Die Vorhersagbarkeit beschreibt das Ausmaß in dem der Entwickler das Ergebnis beeinflussen oder auch nur voraussagen kann, vor allem bei den modell-basierten Ansätzen ist aufgrund der automatischen Übersetzung und Anpassung nicht immer transparent zu welchem Ergebnis eine Designentscheidung führt.

Wie bereits oben gezeigt, hat sich in der Praxis heute der Einsatz von graphischen Interface Buildern durchgesetzt. Um die plattformübergreifende Entwicklung zu unterstützen, ohne die Schwelle zu erhöhen, die Decke zu senken oder die Vorhersagbarkeit zu verschlechtern, erscheint ein direkt-manipulativer Ansatz, basierend auf dem Konzept des Interface Builders daher am geeignetsten.

Angeichts der Komplexität, welche die Entwicklung plattformübergreifender Benutzerschnittstellen aufgrund der vielfältigen Randbedingungen und zu berücksichtigenden Parameter kennzeichnet stellt sich die Frage, ob herkömmliche Entwicklungsumgebungen ausreichende Unterstützung bieten. Sollen existierende Entwicklungswerkzeuge den Entwickler effektiv unterstützen können, müssen sie über den Status des Werkzeuges hinaus gehen und Assistenz bieten. Anstatt eines Entwicklungswerkzeuges soll dem Designer plattformübergreifender Oberflächen ein Entwicklungsassistent an die Seite gestellt werden.

Bereits im vorangegangenen Abschnitt 2.3.3 wurde die Bedeutung der Assistenz für die Zugänglichkeit und Benutzbarkeit interaktiver Systeme hervorgehoben. Wurde dort in erster Linie die Unterstützung behinderter Nutzer betrachtet, steht nun eine andere Benutzergruppe, nämlich die Benutzer des Entwicklungswerkzeugs als interaktivem System im Zentrum der Betrachtung.

Wandke [Wan05] schildert die Fähigkeit des Informationsaustausches zwischen Mensch und Maschine als wesentliches Unterscheidungskriterium zwischen bloßem Werkzeug und Assistenz.



Abbildung 2.22: Assistenz kann die Fähigkeiten des Nutzers erweitern (nach Wandke [Wan05]).

Soll das Ziel sein, den Entwickler bei der Gestaltung effektiv zu unterstützen, muss der Informationsaustausch zwischen dem System und dem Menschen verbessert werden und der Zugang zu Funktionen gewährleistet werden, die der Zielerreichung des Nutzers (hier also des Entwicklers) dienen. Abbildung 2.22 zeigt, wie Assistenz dazu dienen kann die Fähigkeiten des Nutzers zu verbessern. In einer Taxonomie der Assistenzmethoden wird beschrieben, durch welche Arten der Assistenz die

verschiedenen Phasen der Mensch-Maschine Interaktion (*Gulf of Execution and Evaluation*) nach Norman [ND86, Nor88] unterstützen können.

Methoden zur Unterstützung der Entwicklung plattformübergreifender Benutzerschnittstellen stellen eine Form der Entwicklungsassistenz dar, welche in allen Ebenen der Mensch-Maschine-Interaktion ansetzen muss, und damit sowohl die Prozesse der Motivation, Wahrnehmung, Information, Entscheidung, Ausführung und Evaluation von Designentscheidungen unterstützt.

Im Vergleich zur Entwicklung für eine Plattform steht man bei der Entwicklung von Benutzungsoberflächen für verschiedene Plattformen vor einer Vielzahl neuer Probleme. Deklarative oder modellbasierte Beschreibungssprachen, Plattformübergreifende Toolkits oder andere Ansätze, wie sie in Abschnitt 4 beschrieben wurden, bieten zum Teil konzeptionelle Lösungsansätze für diese Probleme. Die praktische Unterstützung des Entwicklers für solche Systeme ist, wie im folgenden Abschnitt gezeigt werden wird, noch immer nicht ausreichend, um diese Ansätzen einer breiteren Anwendung zu öffnen.

2.4.3 Komplexität

Im Gegensatz zu den heute üblichen Methoden, bei denen eine konkrete Darstellung meist anhand sog. Graphischen User Interface Buildern direkt-manipulativ erzeugt wird, wurde bereits im vorangehenden Abschnitt erläutert, dass plattformübergreifende Ansätze bislang in den meisten Fällen über geringe Werkzeugunterstützung verfügen. Ein großer Teil der Entwicklungsarbeit steckt hier in der Erstellung abstrakter Spezifikationen und in der Kontrolle deren Umsetzung auf den verschiedenen Endgeräten. Wendet man die Definition der Komplexität nach Rogers in Bezug auf den plattformübergreifenden Entwicklungsprozess heute an, dann wird klar, dass die Komplexität heutiger Systeme zweifellos zu deren größten Schwächen gehört.

“Complexity is the degree to which an innovation is perceived as difficult to understand and use. Some innovations are readily understood by most members of a social system; others are more complicated and will be adopted more slowly. New ideas that are simpler to understand are adopted more rapidly than innovations that require the adopter to develop new skills and understandings.”[Rog97]

Abbildung 2.23 zeigt den Übersetzungsprozess, welcher bei der plattformübergreifenden Entwicklung von einer abstrakten Spezifikation hin zur konkreten Darstellung notwendig ist. Diese generelle Aufteilung in Schichten wird aufgrund der Notwendigkeit der Abstraktion von von Plattform- und Darstellungsspezifika notwendig. Aufgabe geeigneter Werkzeuge muss es sein diese komplexen Prozesse und die damit verbundenen, nachfolgend aufgeführten, Problematiken vor dem Entwickler zu verbergen und nur bei Bedarf zugänglich zu machen.

Der Übersetzungsprozess von einer abstrakten Spezifikation hin zu der konkreten Darstellung macht einige Transformationsschritte notwendig, bei denen sowohl Information verloren gehen kann, als auch spezifische Information ergänzend in die abstrakte Beschreibung aufgenommen werden kann. Dieser Prozess der Übersetzung führt zu folgenden Problemen, welche bei der Entwicklung von plattformübergreifenden Anwendungen zu einer, im Vergleich zu herkömmlichen Vorgehen deutlich höheren Komplexität führt, die Probleme des *Mapping*, der *Reversibilität*, der *Optimierung* und der *Konsistenz*.

Als *Mapping-Problematik* [PE99, CLC04b] bezeichnet man jene Probleme, welche bei der Konkretisierung der abstrakten Spezifikation zu Darstellung auftreten. In modellbasierten Ansätzen kann diese Mapping Problem zwischen mehreren Modellierungsebenen auftreten bei deklarativen Beschreibungssprachen in der Regel nur bei der Umsetzung der abstrakten Beschreibung in eine Darstellung. Wird dieses Mapping durch Regeln unterstützt oder auf Basis von automatisierten Prozessen durchgeführt, können systematische oder spontane Fehler auftreten. Ein Mapping wird auf verschiedenen Ebenen angewandt, je nachdem wie komplex das Modell ist. So kann im Falle von XI ML (s. 4.3.3.4) ein Mapping-Fehler bei der Abbildung der Aufgaben auf des Dialogmodell oder bei der Abbildung der Domämentypen auf Interaktoren oder bei der Abbildung des Dialoges in ein Präsentationsmodell mit konkreten Interaktoren und der Verteilung in verschiedene Dialoge. In diesem Beispiel stehen also drei mögliche Mapping-Fehler in kausal-verkettetem Zusammenhang.

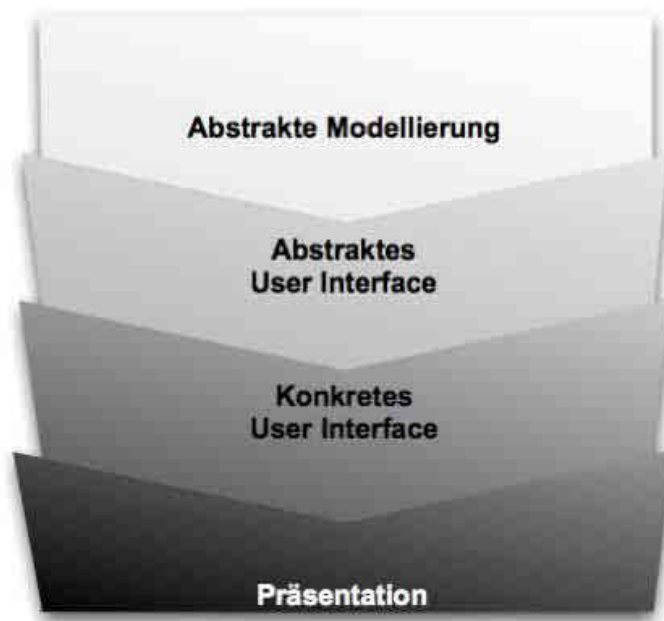


Abbildung 2.23: Darstellung der grundsätzlichen Vorgehensweise bei der Entwicklung plattformübergreifender Oberflächen.

Als *Reversibilitätsproblematik*, auch *Round-Trip-Problematik* [BBSS02], werden Probleme bezeichnet, welche die Rückführbarkeit oder Rekonstruierbarkeit der ursprünglichen Abstraktion aus konkreteren Modellen der Interaktion oder der Darstellung selbst betreffen. Diese Problematik ist eine Invertierung der Mapping-Problematik, da auch hier die Abbildungen zwischen den Abstraktionsebenen ursächlich sind. In der Regel stellen Konkretisierungen einen Informationsverlust dar, sei es weil die konkretere Darstellung nur einen Teilaspekt der abstrakten Beschreibung benötigt, sei es weil ein Teil der abstrakten Beschreibung lediglich der Steuerung der Transformation dient. Diese verlorene Information zu rekonstruieren ist bei der Rückführung notwendig und leider nur bedingt möglich. Stehen auf der abstrakten Seite mehrere Modelle, die in eine konkretere Beschreibung konvergieren ist im Umkehrschluss eine Modifikation in der konkreten Form nur schwer einem der abstrakteren Modelle zuzuordnen.

Als *Optimierungsproblematik* soll eine der übergreifendsten Problematiken der plattformübergreifenden Entwicklung von Benutzungsoberflächen beschrieben werden, der Konflikt zwischen Abstraktion und optimaler Anpassung.

“One of the central ideas behind model based tools is to achieve balance between detailed control over the design of the UI and automation.” [FAPQA04]

Die Ableitung der plattformspezifischen konkreten Darstellung geschieht auf Basis der plattformunabhängigen abstrakten Beschreibung oder Modellierung, welche aufgrund genereller (anwendungsunabhängiger) Abbildungsregeln durchgeführt wird. Die abstrakte Beschreibung muss also alle Informationen enthalten, welche für eine optimale Anpassung notwendig sind. Die Abbildungsregel wiederum muss so spezifisch sein, dass sie zu einer optimal angepassten Darstellung führt. Der Widerspruch liegt hier also offensichtlich in der Forderung nach Abstraktion der Beschreibung vs. Optimierung der Darstellung. Worin äußert sich dieser Konflikt? Zwei Beispiele sollen hier genannt werden:

Das *Design von Benutzungsoberflächen* verliert durch die Abstraktion an Gestaltungsmöglichkeiten. Der Designer hat wenig Möglichkeiten auf spezifische Besonderheiten einzelner Endgeräte einzugehen ohne dass dies Änderungen sich auf alle Darstellungen auswirken. Die generalisierten Abbildungsmechanismen setzen die spezifizierten Beschreibungen i.d.R. viel mehr im Sinne des kleinsten

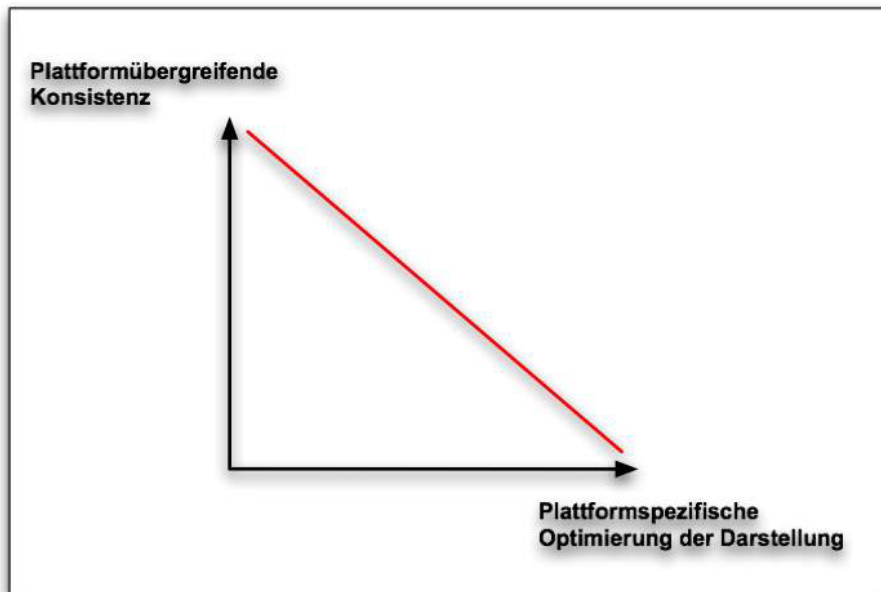


Abbildung 2.24: Die Konsistenz einer Darstellung wird durch die plattformspezifische Optimierung reduziert.

gemeinsamen Nenners als im Hinblick auf die bestmögliche Darstellung um, was häufig zu qualitativ wenig ansprechenden Darstellungen führt [PE99, GZ04].

Die *Optimierung in Abhängigkeit der adressierten Endgeräte* ist ebenfalls ein Problem bei der generalisierten Abbildung. Steht bereits zur Designzeit fest, dass nur eine bestimmte Untermenge der möglichen Plattformen bedient werden soll, kann die Gestaltung u.U. zu besser optimierten UI führen, da z.B. auf bestimmte Einschränkungen keine Rücksicht genommen werden muss. Auch zur Laufzeit kann eine multimodale Abbildung u.U. optimierte Abbildungsregeln verwenden wenn die Schnittmenge der beteiligten Darstellungen eine größere ist.

Modellbasierte Ansätze entgegnen dieser Problematik meist durch eine feinere und aufwändigere Modellierung wobei die zusätzlichen Modelle Informationen über die Plattform- oder Anwendungsspezifika enthalten, was aber im selben Maße die Modellierung verkompliziert und die Mapping-Problematik verschärfen.

Eine weiteres Problem bei der Erzeugung von plattformübergreifenden Benutzungsschnittstelle liegt in der Gewährleistung der *Konsistenz der Darstellung*. Unter Konsistenz einer Anwendung versteht man die Übereinstimmung der Benutzerführung, Darstellung und anderer Eigenschaften einer Benutzerschnittstelle. Man unterscheidet die innere Konsistenz, d.h. die Übereinstimmung innerhalb einer Anwendung oder innerhalb Anwendungen eines Herstellers, bzw. zwischen Anwendungen auf demselben Gerät und die externe Konsistenz, d.h. die Konsistenz zwischen Anwendungen verschiedener Hersteller, oder auf verschiedenen Plattformen [Shn98]. Die Konsistenz wiederum steht im Kontrast zur Optimierungs-Problematik, denn eine optimale Anpassung an die einzelne Plattform kann in der Konsequenz zu einer Abweichung von der zwischen den Plattformen gemeinsamen Linie führen (s. Abbildung 2.24).

Die Konsistenz einer Anwendung wird von Nielsen als eine der wichtigsten Eigenschaften bezüglich der Benutzbarkeit eines System eingeschätzt.

“One of the most important aspects of usability is consistency in user interfaces. Consistency should apply both within the individual application and across computer systems and even across product families.” [Nie89, S. 1]

Der Aspekt der mentalen Belastung und der Vorteile, welche der Benutzer aus einer konsistenten Darstellung ziehen kann, sind aus diesem Grunde kaum zu unterschätzen (s. Abschnitt 5.4). Die

automatische Generierung von Oberflächen aus einer gemeinsamen Ressource kommt dieser Forderung entgegen mit dem Nachteil, der Optimierungs- und Mapping-Problematik. Eine absolute Lösung scheint hier also nicht möglich, sondern vielmehr eine Optimierung der beiden Anforderungen gegeneinander.

2.4.4 Testbarkeit

Ein zentraler Punkt bei der Adaption neuer Technologien ist zweifellos die Möglichkeit, neue Technologien ohne großen Aufwand anhand eines Beispiels testen zu können und hinsichtlich der Nützlichkeit, Kompatibilität und Komplexität (s. Abschnitt 2.4.1–4.4.1.2) beurteilen zu können. Nicht Umsonst stecken viele Entwickler neuer Technologien zunächst einen nicht unerheblichen Aufwand in die Bereitstellung freier und leicht anwendbarer Testszenarien, Beispiele, Demonstrationen oder Evaluationsversionen ihrer Produkte. Rogers bezeichnet dies als die Testbarkeit (engl. *trialability*) einer Innovation.

“Trialability is the degree to which an innovation may be experimented with on a limited basis. New ideas that can be tried on the installment plan will generally be adopted more quickly than innovations that are not divisible. An innovation that is trialable represents less uncertainty to the individual who is considering it for adoption, who can learn by doing.” [Rog97]

Wie Trætteberg [Tra04] anmerkt (s. Abschnitt 2.4.1), ist der Einsatz plattformübergreifender Anwendungsoberflächen bislang mit erheblichem Lernaufwand verbunden, so dass ohne geeignete Werkzeuge für einen interessierten Anwender eine Nutzung mit erheblichem Zeitaufwand verbunden ist. Der Einsatz der meist akademischen Ansätze, die sich zu allem Überfluss in eine kaum überschaubare Menge von konkurrierenden Lösungen aufspaltet (s. Abschnitt 4), ist in der Regel mit großem Einarbeitungsaufwand verbunden, der häufig einer sehr begrenzten Anwendbarkeit gegenübersteht. Wie Molina [Mol04] kritisch anmerkt, ist die Verwendung von Standards wie UML hier leider noch kaum üblich, was die Einarbeitung zusätzlich erschwert.

Fasst man es zusammen, leidet die Testbarkeit heute verfügbarer Ansätze unter der folgenden Problemen:

- *Werkzeuge*: Entwicklung häufig nur eingeschränkt mit Werkzeugunterstützung.
- *Lernaufwand*: Methoden und Syntax meist proprietär und nicht auf Standards basierend.
- *Auswahl*: Anzahl der existierenden Ansätze kaum zu überschauen, welcher Ansatz ist relevant?
- *Funktionalität*: Meist beschränkte Funktionalität der Ansätze durch Fokus auf bestimmte Szenarien. Erweiterbarkeit problematisch.
- *Software*: meist nicht in Produktstadium, vielmehr akademisch. Installation erfordert Konfigurationsaufwand.

2.4.5 Sichtbarkeit

Wie soll sich Innovation verbreiten, wenn sie nicht sichtbar ist? Was sollte die Motivation sein, sie einzusetzen, wenn es dadurch zu keiner, auch für den Kunden sichtbaren, Verbesserung kommt? Zwei Aspekte der Sichtbarkeit (engl. *Visibility*), bzw. Beobachtbarkeit (engl. *Observability*), welche Rogers als wichtige Voraussetzung für die Verbreitung von Innovationen erachtet.

“Observability is the degree to which the results of an innovation are visible to others. The easier it is for individuals to see the results of an innovation, the more likely they are to adopt it. Such visibility stimulates peer discussion of a new idea, as friends and neighbors of an adopter often request innovation-evaluation information about it.” [Rog97]

Die Frage, welche dahinter steht ist die: *Lohnt es sich für den Anbieter, diese Innovation als Verkaufsargument anzuführen?* Bezüglich der modellbasierten Entwicklung drückt sich Trætteberg

[Tra04] folgendermaßen aus: „*I have never seen an application with a 'developed using models' or 'models executed here' sticker or ad [...]*“. Plattformübergreifende Methoden agieren, wenn überhaupt derzeit noch völlig transparent für den Endkunden, zumindest aber ohne spürbare Verbesserung für diesen. Es ist klar, dass das Potential, welches in der plattformübergreifenden Entwicklung auch für den Benutzer steckt geweckt werden muss. Beide Seiten der Nutzung dieser Methoden müssen profitieren, bevor eine ausreichende Nachfrage entsteht um den notwendigen Innovationsdruck aufzubauen. Aus diesem Grund ist die Fokussierung auch der Interessen und Bedürfnisse des Benutzers zentraler Bestandteil dieser Arbeit (s. Abschnitte 2.2 f.).

2.4.6 Zusammenfassung

In diesem Kapitel wurden anhand der Kriterien des *Diffusion of Innovations* Modells von Rogers [Rog97] zentrale Anforderungen an den Entwicklungsprozess plattformübergreifender Anwendungsoberflächen abgeleitet. Die Hauptaugenmerk der hier angestellten Betrachtungen lag dabei auf dem Aspekt der Umsetzung solcher Methoden in einem bereits existierenden Umfeld unter Berücksichtigung der Anforderungen von User Interface Designern und Anbietern solcher Lösungen sowie der Endbenutzer der so entwickelten Lösungen und verfolgt dabei einen klar benutzerzentrierten Ansatz. Aus den oben ausgeführten Überlegungen lassen sich folgende Kern-Anforderungen an den Entwicklungsprozess zusammenfassen:

Erlernbarkeit und Verwendung von Standards: Wenn möglich sollte eine zu entwickelnde Lösung auf existierenden Standards aufbauen, welche in der Domäne bereits Verwendung finden.

Benutzbarkeit der Werkzeuge: Werkzeuge welche den Entwickler bei komplexen Aufgaben unterstützen sollen sollten möglichst intuitiv bedienbar und leicht erlernbar sein. GUI Builder und WYSIWYG

Einbindung in existierende Entwicklungsmethoden: Methoden, welche heute in der Softwareentwicklung Verwendung finden sollten weiterverwendet und möglichst einfach kombiniert werden können. OO-Entwicklung keine neue Sprache, UML...

Transparente Behandlung der bekannten Problematiken: Probleme wie Mapping, Reversibilität, Optimierung und Konsistenz sollten für den Entwickler transparent automatisch gelöst werden. Besteht jedoch die Notwendigkeit soll die Möglichkeit der Anpassung gegeben sein.

Erweiterbarkeit und Anpassbarkeit: Die zu entwickelnde Lösung soll nicht auf einen bestimmten Anwendungsfall beschränkt sein, sondern möglichst einfach erweiterbar und anpassbar sein.

Testbarkeit: Die Schwelle, die vor der Verwendung zu Testzwecken steht sollte durch eine einfache Anwendbarkeit, bzw. die Verfügbarkeit geeigneter Beispiele erleichtert werden.

Sichtbarkeit: Die Lösung sollte zu einem klaren Nutzen auch für den Endbenutzer führen und durch die so gewonnene Attraktivität der Anwendung einen Mehrwert für den Hersteller darstellen.

2.5 Technische Anforderungen

Die Entwicklung und Konzeption eines Systems zur Darstellung und Entwurf von plattformunabhängigen Benutzungsoberflächen ist aufgrund der Aufgabenstellung selbst bereits bezüglich einiger Anforderungen festgelegt. So sind Konzepte zur Berücksichtigung der Anwendung auf verschiedenen Plattformen, d. h. Plattformunabhängigkeit der Implementierung, und die Anpassbarkeit der Ressourcenanforderungen auf Endgeräte verschiedener Leistungsmerkmale, d. h. Skalierbarkeit genauso wie die kontextadaptive Erweiterbarkeit von Interaktionsmitteln und Funktionen z.T. bereits implizit in der Aufgabenstellung enthalten. Im folgenden Abschnitt werden diese Anforderungen im einzelnen erläutert werden.

2.5.1 Plattformunabhängigkeit

Bei der Konzeption eines plattformübergreifenden Systems zu Darstellung und Entwicklung von Benutzungsoberflächen muss ein geeigneter Ansatz zur Realisierung der Plattformunabhängigkeit sowohl der Spezifikation der Benutzungsschnittstelle als auch zur Implementierung der Anwendungslogik gefunden werden.

Die existierenden Lösungen zur Spezifikation von Benutzungsoberflächen werden in Abschnitt 4 diskutiert und beurteilt. Der Schwerpunkt dieser Arbeit liegt klar auf diesem Aspekt. Die plattformunabhängige Spezifikation der Benutzungsschnittstelle wird hier auf der konzeptionellen Ebene beleuchtet und exemplarisch umgesetzt werden. Den Umfang dieser Arbeit sprengen würde die ausführliche Berücksichtigung verschiedener Ansätze der plattformunabhängige Programmierung allgemein. Deshalb sollen hier nur kurz einige existierenden Ansätze vorgestellt werden.

Zahlreiche Ansätze zur Realisierung plattformunabhängiger Programmiersprachen wurden in der Vergangenheit entwickelt, wobei der vorherrschende Lösungsansatz auf dem Konzept von Interpretierten Sprachen oder Zwischen-Codes basiert.

Interpretierte Sprachen (z.B. BASIC [X3.87], ECMA-Script [ECM99] und Python [Ros04]) werden meist in Form von Skripten im Textformat gespeichert und erst zur Laufzeit von einem sog. Interpreter in Maschinensprache umgesetzt. Der Code bleibt damit stets auf beliebigen Plattformen ausführbar, solange für die spezifische Plattform eine Implementierung eines Interpreter-Programms existiert. Nachteil dieses Ansatzes ist vor allem die Ausführungsgeschwindigkeit, da der Interpreter das Skript vor der Ausführung zunächst einliest, analysiert, übersetzt. Die zeitintensive Umsetzung des Skriptes, d.h. des Quellcodes findet also zur Laufzeit statt. Mehrfach durchlaufene Programmteile werden dabei meist auch mehrfach übersetzt. Alternativ hierzu existieren sog. Just-In-Time (JIT) Compiler welche die Quellcodes nur einmal übersetzen und den übersetzten Code dann zwischenspeichern, optimieren und bei wiederholter Ausführung aus dem Zwischenspeicher laden können. Dies verbessert die Ausführungsgeschwindigkeit erheblich.

Zwischen-Codes (z.B. Sun Microsystems Java [Kru04b] oder Microsoft .Net [Pro02]), bei Java Bytecode, bei .Net Microsoft Intermediate Language (MSIL) genannt, sind das Produkt eines Übersetzungsprozesses eines speziellen Compilers in eine plattformübergreifende Zwischensprache. Der Quellcode wird also nicht direkt in Maschinencode übersetzt, sondern in ein Zwischenformat, welches dann zur Laufzeit von einer sog. virtuellen Maschine (bei .Net Common Language Runtime (CLR) genannt) interpretiert. Hier ist es dann die, für jede Plattform gesondert zu implementierende, virtuelle Maschine welche diesen Code in plattformspezifische Anweisungen übersetzt und die Ausführung verwaltet und optimiert. Der plattformunabhängige Zwischencode ist im Vergleich zum Maschinencode sehr kompakt, da er auf die vordefinierten Funktionen der virtuellen Maschine zugreifen kann. Auch bei Bytecode ist die Ausführungsgeschwindigkeit aufgrund der Interpretation zur Laufzeit vermindert und es werden JIT-Compiler eingesetzt um diese zu verbessern.

Die Ansätze welche auf Bytecode basieren haben sich in den letzten Jahren — nicht zuletzt aufgrund des erheblichen Fortschritts der Entwicklungen von Sun Microsystems Java sowie durch die Einführung des Microsoft .Net Frameworks — in vielen Bereichen durchgesetzt. Neben den Vorteilen der einfachen Portierbarkeit des Bytcodes haben sich Mechanismen wie die automatische Speicherverwaltung und die Möglichkeit der Reflektion, d.h. der Analyse und dynamischen Erzeugung von Klassen und Instanzen zur Laufzeit als sehr mächtig erwiesen.

Virtuelle Maschinen sowie Spezifikationen von reduzierten Sprachversionen existieren sowohl für Java als auch für das .Net Framework für verschiedene Endgeräte. Die Entwicklung ist hier bereits in großem Maße von der industriellen und marktseitigen Anforderung nach der einfachen Entwicklung von Anwendungen für die verschiedenen Endgeräte bestimmt und wird aus diesem Grunde von den Herstellern vorangetrieben.

2.5.2 Skalierbarkeit

Mit Skalierbarkeit wird in diesem Zusammenhang die Fähigkeit des Systems bezeichnet, sich an die Leistungsfähigkeit der verwendeten Geräte und an die zur Verfügung stehenden Ressourcen anzu-

passen. Da verschiedene Plattformen in der Regel über verschiedene Leistungsmerkmale (z.B. Arbeitsspeicher, Prozessorleistung, etc.) verfügen, muss ein System, welches auf verschiedenen Plattformen lauffähig sein soll, in der Lage sein in Bezug auf den Ressourcenverbrauch zu skalieren.

Anpassungsparameter sind hierbei in erster Linie die Darstellungsqualität, deren Verminderung den Ressourcenverbrauch reduzierend kann und somit z.B. auf einem Mobilgerät auf Schatten und 3D-Effekte verzichtet werden kann, wohingegen der Benutzer eines Desktop-PC derartiges heute erwartet. Weiterhin kann bei der Anwendung von plattformspezifischen Anpassungsprozessen (Adaption) zugunsten von Standardprozeduren anstatt Echtzeitberechnungen der Ressourcenverbrauch reduziert werden. Auch der logische Aufbau der Implementierung, die Verwendung von vorkompilierten Komponenten vs. Interpretieren kann den Ressourcenverbrauch vermindern.

Das zu entwickelnde System sollte die Möglichkeit der Skalierbarkeit berücksichtigen und verschiedene Möglichkeiten der plattformspezifischen Anpassung des Ressourcenverbrauchs in die Konzeption mit einbeziehen.

2.5.3 Dynamische Erweiterbarkeit

Mit dynamischer Erweiterbarkeit wird die Möglichkeit des Systems bezeichnet, neue Komponenten dynamisch zur Laufzeit oder zumindest bei der Initialisierung ohne aufwändige Konfiguration einzubinden oder zu entfernen. Diese Anforderung entsteht vor allem aus den, in Abschnitt 2.3 geschilderten Anforderungen an die Benutzbarkeit, wobei eine Anpassung an den Benutzer, dessen Kontext und Aufgabe als Eigenschaft eines benutzbaren Systems betrachtet wird.

Muss das System an den Benutzer und dessen Interaktionsstil und Möglichkeiten angepasst werden sollte das System in der Lage sein neue Interaktionsmittel und Ausgabemodalitäten (s. Abschnitt 2.3.3), spezifisch an den Benutzer angepasst zu integrieren.

Geht man von einem System aus, welches als Zugang zu einer veränderlichen Umgebung (s. Abschnitt 1.1.2) dient, besteht eine weitere Anforderung in der Fähigkeit Dienste, welche dynamisch aus der Umgebung angeboten werden und auch wieder verschwinden können (z.B. aufgrund eines Ortswechsels) zu erkennen und deren Benutzbarkeit für den Benutzer des Systems zu gewährleisten, d.h. also deren Benutzungsoberfläche zur Verfügung stellen zu können.

2.5.4 Standardkonformität

Wie bereits in Abschnitt 2.4 motiviert, stellt die Verwendung von Standards, bzw. die Standardkonformität eine wichtige Anforderung an ein neu zu entwickelndes System dar. In jedem Fall sollte vor einer Neuentwicklung von Spezifikationen und Techniken die Verfügbarkeit kompatibler Standards abgeklärt werden. Vorteile hiervon sind, neben der oben angeführten Akzeptanz und verbesserten Erlernbarkeit, die Möglichkeit auf existierende Werkzeuge und Prozessmodelle zurückzugreifen. Über den Beitrag zu offiziellen Standardisierungsbemühungen ist eine schnellere Verbreitung der Lösung gegeben wie auch die Möglichkeit der kritischen Diskussion, welche zu einer konstruktiven Verbesserung der Lösung führen kann.

Verfügbare Standards und Lösungen auf dem Gebiet der plattformübergreifenden Benutzungsoberflächen werden in erster Linie in Abschnitt 4 besprochen. Offizielle Standards bzw. in der Standardisierung befindliche Ansätze seien hier nochmals kurz zusammengefasst:

- *W3C Hypertext Markup Language (HTML / XHTML)* — Markupsprache zur Beschreibung von Inhalten im WWW. Wird häufig als UI für serverbasierte Anwendungen verwendet, aufgrund der Anzeige in Browsern und dem hier implementierten Hypertext Transfer Protokolls sowie aufgrund der laxen Einschränkung der Syntax aber in der Form wenig geeignet als Format für Anwendungs-UI.
- *W3C Extensible Markup Language (XML)* — Basisformat für viele Datenformate, die auf XML aufsetzen. Hauptvorteile sind die Plattform- und Anwendungsunabhängigkeit, da textbasiert offen und einfach zu verarbeiten, die Verfügbarkeit weiterführender Anwendungen (z.B. XHTML, UIML, u.a.) und Werkzeuge.

- *W3C XForms* — Ein Standard für die nächste Generation von Web-Formularen zur Einbettung in XHTML, spezifiziert UI-Elemente auf einer relativ abstrakten Ebene unter Trennung von Darstellung, Datenmodell und Inhalten.
- *OASIS User Interface Markup Language (UIML)* — Eine derzeit in der Spezifikationsphase befindliche Beschreibungssprache für User Interfaces.
- *Cascading Stylesheets (CSS)* — Ein Standard zur Beschreibung von Styleinformationen, v.a. zu Einbettung in HTML.

Insbesondere das World Wide Web Consortium hat sich in diesem Bereich bezüglich der Standardisierung von offenen Webtechnologien und Technologien des Informationsaustausches in offenen System hervor getan und verschiedene Bemühungen zu einer Vereinheitlichung der Ansätze beigetragen. Im Rahmen der *Device Independence Workgroup (DIWG)* [Wor01] und der *Multimodal Interaction Activity (MMI)* [Wor02] wird derzeit versucht, geeignete Formate und Ergänzungen existierender Standards zu erarbeiten. Schwerpunkt ist hierbei die Beschreibung von Geräten und Präferenzen (CC/PP)[KRW⁺04] gerätespezifische Selektion von Inhalten (DSelect)[LM04], multimodale Annotation von HTML-Inhalten (EMMA)[CDJP04] und Integration von Sprache und Gestik.

All diese Anstrengungen stehen in unmittelbarem Zusammenhang mit den existierenden Standards des WWW und adressieren weniger die Domäne der geräteunabhängigen Anwendungsoberflächen. Dennoch sollte im Rahmen der Konzeption und Entwicklung darauf geachtet werden, existierende Standards weitestgehend zu berücksichtigen.

2.5.5 Zusammenfassung

In diesem Kapitel wurden die zentrale technischen Anforderungen an das zu entwickelnde System formuliert. Diese lassen sich wie folgt zusammenfassen:

Plattformunabhängigkeit: Das System sollte so gestaltet sein, dass es auf verschiedenen Plattformen lauffähig ist und das System unabhängig von dem verwendeten Gerät anwendbar ist.

Anpassung an Systemressourcen: Das System sollte bezüglich der benötigten Systemressourcen variabel sein, so dass es auch auf Geräten mit geringem Speicher und Prozessorleistung lauffähig ist.

Dynamische Konfiguration: Komponenten sollten zur Laufzeit in das System zu integrieren sein, bzw. ohne großen Aufwand eingebunden werden können.

Verwendung von Standards: Technische Lösungen und Formate sollten wo möglich auf existierenden Standards aufsetzen.

2.6 Zusammenfassung

In diesem Abschnitt wurde zunächst anhand einer explorativen Benutzerstudie versucht, zentrale Anforderungen des Benutzers von plattformübergreifenden Anwendungen zu erkennen und zu sammeln. Die weitere Analyse von Anforderungen wurde aus verschiedenen Perspektiven vorgenommen um so ein möglichst vollständiges Bild zu erhalten. Berücksichtigt wurden hierbei die Anforderungen an die Benutzbarkeit, in welche die Ergebnisse der Studie einfließen, die Anforderungen an den Entwicklungsprozess, welcher vor allem als Teil des Prozesses der Softwareentwicklung auch wirtschaftliche und marktpolitische Aspekte berücksichtigt und den technischen Anforderungen, welche vor allem die Aspekte, welche bei der Implementierung relevant werden, beleuchtet. Zwischen den verschiedenen Perspektiven ließen sich Anforderungen identifizieren, welche wiederkehren und somit aus allen Blickwinkeln von Bedeutung sind, diese können zusammengefasst werden.

Aus den Überlegungen, die in diesem Abschnitt angestellt wurden lassen sich somit die im Folgenden beschriebenen Kernanforderungen an das System definieren. Diese werden die Grundlage für die Konzeptionsphase darstellen.

Plattformübergreifende Benutzbarkeit: Ein wichtiger Aspekt der plattformübergreifenden Darstellung und Entwicklung von Benutzungsoberflächen ist die Benutzbarkeit der erzeugten Oberflächen. Neben den gängigen Anforderungen an die Benutzbarkeit [ISO99a, ISO01, ISO99d] steht vor allem die Wissenskontinuität und die plattformübergreifende Benutzbarkeit im Vordergrund. Der Benutzer sollte Wissen, welches er bei der Benutzung des Systems auf einer Plattform erworben hat auf einer anderen Plattform anwenden können.

Entwicklungsunterstützung: Die Werkzeuge welche den Entwickler bei komplexen Aufgaben unterstützen sollen sollten möglichst intuitiv bedienbar und leicht erlernbar sein. Probleme wie Mapping, Reversibilität, Optimierung und Konsistenz sollten für den Entwickler transparent automatisch gelöst werden. Besteht jedoch die Notwendigkeit soll die Möglichkeit der Anpassung gegeben sein. Die Unterstützung durch automatische Prozesse zur Erzeugung der Darstellung soll nicht zu einer Entmündigung des Designers, oder zu wenig attraktiven Endprodukten führen. Auf der anderen Seite muss die Unterstützung so weit gehen, dass ohne großen Konfigurationsaufwand Lösungen zu erzeugen sind.

Adaptivität: Das System sollte in der Lage sein, sich auf die Bedürfnisse des Benutzers — in einer bestimmten Aufgabe auf einem bestimmten Gerät und in einem bestimmten Umfeld — anzupassen. Die Anpassung an die Systemeigenschaften, Darstellungsfähigkeiten usw. sind zentrale Anforderungen und Teil der Aufgabenstellung. Als Teil der Kontextadaptivität sollte die Darstellung der Benutzeroberfläche in verschiedenen Modalitäten in der für den Benutzer günstigsten Konstellation möglich sein. Wichtige Dimensionen der Kontextadaptivität sind hierbei:

- Benutzer: Körperliche und geistige Fähigkeiten, Modalitäten, Präferenzen
- Gerät: Ressourcen und Leistung, Darstellungsmöglichkeiten, Interaktionsmittel
- Umgebung: Umweltgeräusche, Licht, verfügbare Dienste

Verwendung von Standards: Der Erfolg einer neuen Lösung hängt in erster Linie von deren Anwendbarkeit ab und von der Schwelle, welche vor der Erprobung, dem Erlernen und der Einführung steht. Um diese niedrig zu halten und die Weiterverwendung existierender Lösungen, Prozesse und Vorwissen zu ermöglichen sollten existierende Standards und gängige Entwicklungsmethoden weitestgehend unterstützt werden.

3 Grundlagen

In diesem Abschnitt werden die grundlegenden Konzepte und Theorien eingeführt, welche zum weiten Verständnis dieser Arbeit notwendig sind. Im weiten Verlauf dieser Arbeit wird an den entsprechenden Stellen auf die Grundlagen-Kapitel verwiesen werden.

Dieses Kapitel enthält zunächst eine Einführung in das Phänomen der Übertragbarkeit von Wissen und der damit verbundenen Frage der Repräsentation von Wissen. Der nächste Abschnitt behandelt dann einen weiteren Aspekt der menschlichen Wahrnehmung, das Konzept der mentalen Modelle. Darauf folgend wird die damit verwandte Problematik der Konsistenz von Benutzerschnittstellen und deren Einfluss auf die Benutzbarkeit von Anwendungen diskutiert. Ein zentraler Beitrag dieser Arbeit besteht in der Anwendung dieses Konzeptes auf graphische Anwendungsoberflächen verschiedener Endgeräte. Die Argumentation in diesem Kapitel hat auch zum Ziel, die Notwendigkeit eines solchen Ansatzes zu verdeutlichen.

3.1 Wissenstransfer

It has been claimed that, in comparison with simpler machines like automobiles and copiers, transfer among different kinds of computer systems is relatively difficult (Nakatani, 1983). Perhaps transfer should be considered more explicitly in the design process.

— Singley & Anderson, *Transfer of Cognitive Skill*, 1989

Wie bereits beschrieben, hat sich diese Arbeit zum Ziel gemacht, den Transfer von Wissen und Erfahrung im Umgang mit einer Anwendung auf verschiedenen Endgeräten zu vereinfachen und Methoden zu entwerfen, welche den Entwicklungsprozess verbessern und beschleunigen. Zunächst soll hier auf den Hintergrund und der Stand der Forschung in der Psychologie des Wissenstransfers eingegangen werden.

Die Frage, wie gelerntes Wissen auf neue Situationen transferiert und angewandt wird, beschäftigt die Menschen zumindest seit der Antike. Bereits Aristoteles formulierte eine *Theorie der geistigen Fähigkeiten*, die sich in Eigenschaften wie Beobachtung, Aufmerksamkeit, Unterscheidung und Analyse gliedern. Diese generellen Fähigkeiten sollten durch Übung wie Muskeln gestärkt werden können. Generelle Fähigkeiten können nach dieser Ansicht auf beliebige Inhalte angewendet werden. Die Folgen einer solchen Sicht—die sich bis weit in das 20. Jahrhundert hielt—für die Erziehung waren enorm. Es wurde daraus geschlossen, dass der Lerninhalt selbst mehr oder minder unwichtig sei. Die reine Praxis und die Disziplin des Geistes war es, was mit Fächern wie Latein gefördert werden sollte. Transfer, so die klassische Ansicht sollte innerhalb solcher Fähigkeitsbereiche stattfinden [Bor50].

Als einer der ersten empirischen Psychologen stellte der *Assoziationstheoretiker Thorndike* [TW01] die Theorie der generellen Fertigkeiten und die Doktrin der formalen Disziplin in Frage. Thorndike vertrat die radikale Gegenposition, nach der gelerntes Wissen nur auf spezifische Situationen angewandt werden könne. Der Geist verfüge nicht über generelle Fähigkeiten, sondern besteht aus einer Sammlung spezifischer Handlungsweisen und Assoziationen, die stets mit spezifischen Situationen verknüpft sind [SA89]. Gelernt werden also spezifische *Stimulus-Response*-Paare. Transfer kann demnach nur stattfinden, wenn eine neue Situation zumindest teilweise über identische Elemente verfügt.

Thorndike versuchte mit einer Reihe von Experimenten, zum einen die Theorie der generellen Fähigkeiten zu widerlegen und zum anderen seine Theorie der identischen Elemente zu stützen. Obwohl Thorndike zeigen konnte, dass die Gedächtnisleistung für Worte nicht durch Gedächtnisübungen mit Zahlen verbessert werden konnte [TW01], musste er feststellen, dass sehr wohl ein positiver Transfer bei der Lösung verschiedenartiger algebraischer Gleichungen auftrat [Tho22]. Als einer der größten

Schwachpunkte in Thorndikes Idee stellte sich jedoch die Bestimmung des Begriffs *identisch* heraus, da dieser nicht klar definiert werden konnte.

Thorndikes großer Verdienst liegt in der Widerlegung der Theorie der generellen Fähigkeiten. Die Theorie der identischen Elemente jedoch scheiterte an der mangelnden Möglichkeit zu beschreiben welche Elemente ausschlaggebend sind und wie eine Situation als ähnlich erkannt wird.

Mit dem *Entwicklungspsychologen Piaget* schlug das Pendel der Geschichte wieder auf die Seite der Generalisten. Der große Beobachter Piaget stützte mit seinem Entwicklungsphasenmodell die Ansicht, dass sich die mentale Reife in der Verfügbarkeit genereller kognitiver Fähigkeiten ausdrückt. Die vier Hauptstadien der geistigen Entwicklung treten nach Piaget in einer festen Abfolge auf und sind nicht reversibel [OM95, S. 518ff.]. Piaget entwickelte eine ganze Reihe von diagnostischen Experimenten, welche zur Einordnung in verschiedene Reifestufen genutzt werden können und zum Teil noch heute Verwendung finden.

Transfer kann nach Piaget nur innerhalb der, für eine Phase typischen Fähigkeiten stattfinden, sollte jedoch nicht inhaltspezifisch sein. Allerdings wurde bei Kindern in der so genannten *konkret operationalen Phase* der Entwicklung festgestellt, dass diese das Bewusstsein der Invarianz der Substanz, des Gewichts und des Volumens bei Anordnungs- und Formveränderungen nicht zeitgleich sondern zeitlich verschoben erlernen (*horizontal decalage*). Auch in anderen Aufgaben scheint der Entwicklungsprozess nicht generell, sondern inhaltspezifische Fähigkeiten hervorzubringen. Ein Indiz für die Bedeutung der Inhalte und deren Repräsentation für den Prozess des Lernens und Anwendens.

Thorndikes Theorie der identischen Elemente wurde von den *Gestaltpsychologen* um Wertheimer wieder aufgegriffen. Die Gruppe um Wertheimer, Koffka und Katona wiesen darauf hin, dass die Assoziationstheoretiker höhere Strukturen des Wissens außer Acht gelassen hätten. Sie unterschieden zwischen sinnlosem (assoziativem) und bedeutungsvollem Lernen bei dem aus den einfachen Fakten die wahren Zusammenhänge erkannt werden [SA89]. Ein Beispiel für diese höhere Organisation von Wissen gibt Wertheimer [Wer45] anhand der Wahrnehmung von Musik. Der Mensch nimmt zwei transponierte Versionen eines Musikstückes als identisch wahr, selbst wenn diese keine Note gemeinsam haben, da er das Stück als Ganzes und die Noten in ihrer funktionalen Beziehung wahrnimmt: Das Ganze ist mehr als die Summe seiner Teile.

Transfer sollte in diesem Falle möglich sein, wenn die Grundstruktur, die höhere Organisation des zur Lösung nötigen Wissens gemein ist. Die Erkenntnis dieser höheren Organisation hängt jedoch in großem Maße vom Betrachter ab, davon wie dieser instruiert und welche Vorkenntnisse zur Analyse einer Situation er besitzt. Singley und Anderson [SA89] schließen mit der Feststellung, dass „when considering issues of transfer, we must consider the strategies and representations used by subjects. Some may be more conducive to transfer than others.“

Die Frage nach der Form des Wissens, der Repräsentation der Information im menschlichen Geist rückt somit immer stärker in den Fokus der Betrachtungen. Die Elemente des Wissens zu isolieren ist das eine Problem, den Zusammenhang dieser Elemente zu identifizieren und die Ähnlichkeit zu bestimmen das andere. Aufgrund der guten Kontrollierbarkeit und der klaren Struktur eignen sich insbesondere verbale Stimuli für die gezielte Variation der experimentellen Bedingungen, die Elemente sind hier Worte, die Beziehung bestimmt sich über deren semantische Relationen. Osgood [Osg49] stellte ein Modell auf, welches die Ergebnisse der Transferstudien anhand der Ähnlichkeitsbeziehung zwischen Stimuli und Responses in beiden Listen (erste Liste und Transferliste) in ein dreidimensionales Schema bringt. Auch wenn die so beschriebenen Zusammenhänge zum Teil trivial erscheinen (ähnliche Stimuli und Response-Listen führen zu höherem Transfer) ist der wesentliche Verdienst der, eine erste systematische Ordnung und analytische Beschreibung der Elemente, ihrer Beziehung und des Einflusses auf den Transfer vorgenommen zu haben.

Die Schule der *Programmed Instruction* versuchte diesen analytischen Anspruch auf die komplexe Situation der Klassenzimmer zu übertragen. Die Gruppe um den Instruktionspsychologen Gagné versuchten, Lerninhalte in die hierzu notwendigen Fertigkeiten zu zerlegen und in eine Hierarchie der einander sich bedingenden Fertigkeiten einzuordnen. Einheiten des Transfers waren somit, wie bei Piaget, kognitive Fertigkeiten. Die Beziehung zwischen ihnen stellte sich als eine Teil-Ganzes-Beziehung in einer Hierarchie dar. Ziel des programmierten Lernens ist es, den Lernenden so durch diese Hierarchie zu führen, dass sich die Lerneinheiten entsprechend der Hierarchie anordnen und

die notwendigen Grundkenntnisse vor dem Einführen neuer Inhalte vermittelt werden [Gag66]. Gagné konnte zeigen, dass diese Vorgehensweise generell zu besseren Resultaten führte, der Effekt jedoch in erster Linie bei schwachen Lernern zu beobachten war, nicht bei starken Lernern die auch in der Lage waren Schritte zu überspringen.

Ähnlich dem Phasenmodell Piagets sollten auch hier Fertigkeiten aufeinander aufbauen. Gagné [Gag66] definierte aus diesem Grund zwei Formen des Transfers, den *lateralen Transfer* zwischen ähnlichen Situationen gleicher Komplexität (z.B. zwischen Programmiersprachen), und dem *vertikalen Transfer*, als Beitrag einer weniger komplexen Fertigkeit als Teil einer komplexeren Fertigkeit auf einer höheren Ebene.

Mit der *kognitiven Wende* in der Psychologie der sechziger Jahre und dem Aufkommen der informationstheoretischen Modelle, verlagerte sich das Interesse der Forschung immer stärker auf die elementaren kognitiven Prozesse. Die ganzheitlichen Theorien wurden zugunsten der Erforschung einzelner kognitiver Phänomene zurückgestellt.

Insbesondere in der Frage, wann Menschen von Analogien in Aufgabenstellungen durch Transfer zu profitieren in der Lage sind, beschäftigte die Forschung der siebziger Jahre [GH83, GH80]. Eines der experimentellen Paradigmen, welche in diesem Zusammenhang verwendet wurden, war das *Strahlungsproblem* [Dun45]:

„Stellen Sie sich vor, Sie seien ein Doktor, der einen Patienten mit einem inoperablen Magentumor vor sich hat. Mithilfe einer Strahlentherapie könnten Sie den Tumor zerstören. Wie können Sie den Tumor mit einer ausreichenden Strahlendosis bestrahlen ohne das umliegende Gewebe zu zerstören?“ [Dun45]

Gewöhnlich sind nur 10% der Probanden in der Lage, dieses Problem zu lösen. Die Lösung besteht darin, die Bestrahlung aus verschiedenen Winkeln mit demselben Fokus vorzunehmen. Die Beschreibung einer analogen militärischen Problemstellung (Angriff auf ein Fort, ohne die Gefahr einzugehen, dass alle Truppen auf demselben Weg in einen Hinterhalt geraten), konnte die Lösungswahrscheinlichkeit nur auf 30% heben.

Das Problem besteht aus zwei Teilen: die Person muss sich zum einen erst einmal bewusst werden (*noticing*), dass die eine Analogie besteht, zum anderen muss sie dann in der Lage sein, den Lösungsansatz auf die neue Situation zu übertragen (*mapping*).

In vielen Fragestellungen ist die Analogie ausreichend klar um den Transfer zu bewerkstelligen, wenn das Bewusstsein über diese Analogie vorhanden ist [GH80]. Allerdings hängt in vielen Fällen der Transfer auch von der Fähigkeit zum Mapping ab. Dieses Mapping ist häufig von der Fähigkeit des Probanden abhängig, die Tiefenstruktur des Problems zu erkennen. Diese Fähigkeit hängt wiederum häufig von den Vorkenntnissen der Probanden ab [CGR82].

Eines der bekanntesten experimentellen Paradigmen zur Untersuchung des *Mapping Problems* ist die *Kartenaufgabe* von Wason [Was66]. Probanden erhalten vier Karten, auf denen die Zeichen A, E, 4 und 7 gezeigt werden. Die Karten liegen auf dem Tisch. Der Proband wird gebeten nur die Karten umzudrehen, die notwendig sind, die Behauptung zu überprüfen: „Wenn eine Karte einen Vokal auf einer Seite hat, dann hat sie eine gerade Zahl auf den anderen.“ Nur ca. 4% der Probanden wählten die korrekten Karten, das E und die 7, um die Behauptung falsifizieren zu können. Auch Probanden mit Vorkenntnissen in formaler Logik konnten ihr Wissen nicht auf diese Aufgabe übertragen.

Cheng *et al.* [CHNO86] konnten jedoch zeigen, dass eine natürlichere Fragestellung wie, „wenn eine Person Alkohol trinkt, dann ist sie über 21“ die Leistung deutlich verbesserte. Cheng *et al.* interpretierten dieses Ergebnis dahingehend, dass Probanden aufgrund der alltäglichen Erfahrung ein *Schema* für die Überprüfung von Verboten besitzen, welches hier zum Einsatz komme. Diese Betrachtungen motivierten u. a. Johnson-Laird [JL83] zu der Annahme einer pragmatischen mentalen Repräsentation von Wissen, dem *mental Model* (s. nächster Abschnitt).

Singley und Anderson [SA89] beschreiben den Transfer kognitiver Fertigkeiten bei der Benutzung von Informationstechnik, und hier insbesondere von Textverarbeitungsprogrammen. In einer ganzen Reihe von Studien konnten Singley und Anderson zeigen, dass Transfer (*lateral, vertikal, negativ*) in Form von zwei Arten von Wissen erfolgt: kurzfristig in *deklarativem Wissen* (wie ist etwas) und

langfristig in dem *prozeduralen Wissen* (wie mache ich etwas). „We have resurrected Thorndike’s theory by redefining his identical elements as the units of declarative and procedural knowledge in the ACT* theory.“ Dieses Wissen wird in Form von *Produktionen* (Bedingungs-Aktions-Regeln) repräsentiert. Regel- bzw. Produktionssysteme wie ACT* oder ACT-R bieten zusätzlich die Möglichkeit, anhand von Simulationen empirische Befunde zu überprüfen. Anderson ist es mit der ACT Theorie gelungen, Thorndikes Frage nach den Einheiten des Wissens zu präzisieren und den Begriff der Ähnlichkeit anhand der Kombination von Auslösebedingungen zu beschreiben. Die Frage der Gestaltpsychologen zu der höheren Organisation von Wissen lässt sich hiermit jedoch schwer fassen, so dass die Frage der Repräsentation von Wissen nur unvollständig beantwortet scheint.

Transfer, so scheint es, ist in erster Linie gebunden an die Anwendbarkeit des Wissens auf die Situation. Diese Anwendbarkeit hängt davon ab, wie leicht sich die Analogie erschließt—*noticing*—und wie leicht das Wissen sich dann übertragen lässt—*mapping*. Je genereller dieser Transfer, d. h., je weniger inhaltliche Überlappung zwischen Ursprungs- und neuer Situation existiert, desto größer der Abstraktionsaufwand und desto geringer der Nutzer. Umgekehrt gilt, je spezifischer dieser Transfer, desto stärker die Überlappung, desto größer die Passung und desto größer die Ersparnis für den Menschen [SA89].

Wie diese Betrachtung gezeigt hat, ist die Frage des Transfers von Wissen untrennbar mit der Frage nach der mentalen Repräsentation von Wissen verbunden. Was auf der elementaren Ebene Thorndike als Einheiten, Anderson als Produktionen bezeichnete, was von den Gestaltpsychologen mit dem Begriff des Ganzen und der Frage nach dessen höheren Organisation betrachteten bleibt Grundlage und Ausgangspunkt aller Betrachtungen in der kognitiven Psychologie.

3.2 Mentale Repräsentation

Any chapter entitled “Mental Representation” needs to be subtitled with the comforting words “Don’t Panic!”, like Douglas Adams’ (1984) inter-galactic hitch-hikers’s guide.

— Eysenck & Keane, *Cognitive Psychology*, 1993

Wie Eysenck und Keane [EK93] weiter feststellen, ist die Frage, „[...] how we represent the world *inside our heads* has been one of the big questions in philosophy, psychology, and linguistics for centuries.“ Analog zu den zwei Typen der externalen Darstellung von Informationen, der bildhaften (*piktoralen*) und der sprachlichen (*linguistischen*) Repräsentation, wird in der kognitiven Psychologie von zwei verschiedenen Prozessen der internalen Repräsentation ausgegangen. Neben der Annahme *verteilter* Repräsentationen, die insbesondere die Klasse der konnektionistischen Modelle der Informationsverarbeitung [MR86] hervorgebracht haben, haben die Vertreter der Annahme symbolischer Repräsentationen einen großen Beitrag zur Erforschung kognitiver Phänomene geleistet.

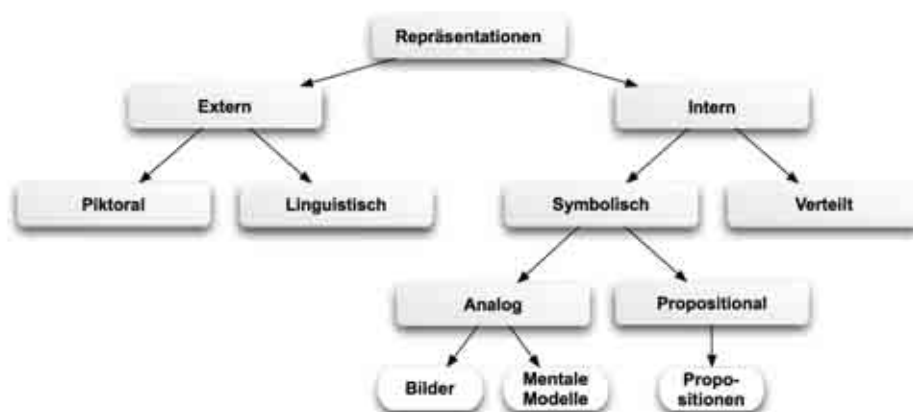


Abbildung 3.1: Klassifikationsschema externaler und mentaler Repräsentationen nach [EK93].

Abbildung 3.1 zeigt ein Übersichtsschema der Klassifikation mentaler Repräsentationen. Die symbolischen (zeichenhaften) Repräsentationen umfassen hierbei die *analogen* (z.B. Bilder, Geräusche,

etc., aber auch mentale Modelle) und die *propositionalen* (abstrakten linguistischen) Repräsentationen. Kosslyn [Kos80, nach [EK93]] nennt vier Aspekte in denen sich linguistische und bildhafte Repräsentationen unterscheiden: *erstens*, linguistische Repräsentationen bestehen aus diskreten, nicht trennbaren Symbolen („B“ kann nicht mehr unterteilt werden, ein Bild schon); *zweitens*, linguistische Symbole sind eindeutig einer Bedeutung zugeordnet („Buch“ hat eine Bedeutung, eine Linie nicht); *drittens*, grammatische Regeln organisieren linguistische Symbole, pikturale Symbole haben keine Grammatik; und *viertens*, linguistische Repräsentationen sind abstrakt und nicht modalitätsspezifisch, pikturale Darstellungen entspringen der visuellen Wahrnehmung.

Analoge Repräsentation Nachdem in der Antike die bildhafte Vorstellung als die Grundfunktion des menschlichen Geistes angesehen wurde, drängte die behavioristische Schule diese Ansicht zunächst zugunsten des propositionalen Ansatzes zurück [EK93]. Paivio [Pai86] konnte mit der *Dual-Coding Theorie* die Annahme einer bildhaften Repräsentation stützen. Paivio postulierte zwei unabhängige, jedoch verbundene repräsentationsspezifische mentale Systeme. Das verbale und das nonverbale System sind jeweils für die Verarbeitung (Enkodierung, Organisation, Speicherung und Abruf) der jeweiligen Repräsentationen zuständig. Das nonverbale System ist spezialisiert auf räumliche, bildhafte Informationen, welche in sog. *Imagenen* geordnet sind. Das verbale System verarbeitet Sprache, abstrakte symbolische Information in sog. *Logogenen*. Logogene und Imagen sind referenziell semantisch verknüpft, so wird der Begriff *Schnee* mit einer bildhaften Vorstellung von Schnee verknüpft.

Diese Annahme wurde in verschiedenen Experimenten gestützt. So konnte z. B. eine bessere Erinnerungsleistung für konkrete (bildlich vorstellbare) Worte gegenüber abstrakten Worten festgestellt werden (s. auch [EK93, S. 214ff.]). Weitere Unterstützung für die These der bildhaften mentalen Verarbeitung konnten z.B. in den Untersuchungen zur *mentalen Rotation* [CS73] (s. auch Abschnitt 7.2.1) und in den Untersuchungen zur *Kartensuche* [KBR78] gefunden werden. Räumliche Zusammenhänge, so die Grundaussage, bleiben auch bei rein mentalen Aufgaben auf visuellem Material bestehen. Die Kritik von Zenon Pylyshyn [Pyl81, Pyl73] richtete sich gegen die Metapher der bildhaften Repräsentation und wird in [EK93, S. 222ff.] dargestellt. Aufbauend auf Pylyshyns Kritik und dem *Dual-Coding Modell* entwickelte Kosslyn [Kos80] ein Modell in dem repräsentationale und analoge Aspekte der Darstellung auf verschiedenen Abstraktionsebenen (räumliches Medium, bildhafte und propositionale Speicherung) verknüpft werden.

Propositionale Repräsentation Die propositionale Repräsentation von Wissen hat die Eigenschaft, dass sie unabhängig von Sprache oder Modalität Wissen auszudrücken in der Lage ist. Sie steht für die basalen mentalen Repräsentationen, welche die Bedeutung der Welt und die grundlegenden Konzepte in der menschlichen Vorstellung auszudrücken in der Lage sind. Eyesenck und Keane [EK93, S. 220] beschreiben die propositionalen Repräsentationen als „[...] a universal, amodal, *mentalese*, the basic code in which all cognitive activities are proposed to be carried out.“

Die Aussage „Ein Buch liegt auf dem Tisch“ drückt sich als Proposition in der Form

AUF(*BUCH*,*TISCH*)

aus, wobei, die Worte nicht die lexikalischen Worte der deutschen Sprache, sondern die *Idee* oder *Bedeutung* dieser Begriffe darstellen. *AUF* wird hierbei als Prädikat bezeichnet.

Aufgrund dieser hohen Abstraktion und der Annahme, dass dies der eigentliche Code des Gehirns ist, lässt sich hierüber schwer ein empirischer Nachweis führen. Propositionale Aussagen wurden daher auch primär zur Formulierung von Modellen der menschlichen Informationsverarbeitung genutzt.

Mentale Modelle Mentale Modelle werden als unvollständige Repräsentationen gesehen, welche sich durch ständigen Abgleich mit der beobachteten Realität verändern und anpassen. Nach Johnson-Laird [JL83] stellt sich die Integration neuer Inhalte in das mentale Modell des Benutzers als ein Übersetzungsprozess dar. Die in der Realität wahrgenommenen Gegebenheiten, werden zunächst in einen sensorischen Zwischenspeicher rein visuell und unverarbeitet aufgenommen. Über

einen ersten Verarbeitungsschritt wird die wahrgenommene Information durch eine *repräsentationale Semantik* in die interne Repräsentation aufgenommen. Neue Inhalte werden zu neuen Repräsentationen, Verknüpfungen zu bekannten Konzepten werden erstellt und Konflikte durch Neuverknüpfungen gegebenenfalls aufgelöst.

Wie ist es zu erklären, dass sich Menschen mit manchen komplexen Systemen problemlos zurechtfinden und zugleich an einfachen Interaktionen scheitern können? Weshalb erscheinen uns manche Anwendungen dauerhaft unverständlich und andere, mit demselben Umfang intuitiv und einfach zu verstehen? Die Forschung auf dem Gebiet der Mensch-Maschine Interaktion und speziell die Untersuchung bei der Benutzung von informationstechnischen Systemen hat versucht, eine Antwort auf diese Frage zu geben, und Methoden zu bieten, die Benutzbarkeit und Verständlichkeit von Computer-Anwendungen zu verbessern.

Ein wichtiger Aspekt hierbei ist der Informationsaustausch zwischen dem System und dem Benutzer. Wie vermittelt das System seine Funktionalität? Welche Vorkenntnisse kann der Benutzer bei der Bewältigung seiner Aufgabe verwenden? Wie kann der Benutzer die neuen Fertigkeiten in sein bisheriges Wissen integrieren?

Auf der einen Seite muss also das System dem Benutzer vermitteln was es kann. Norman bezeichnete in seinem eindrucksvollen Buch *The Psychology of Everyday Things* [Nor88] diesen Prozess als die Vermittlung der *Affordances* (sprich: Möglichkeiten) einer Anwendung. Idealerweise erschließt sich dem Benutzer also aufgrund der Gestaltung des Objektes dessen Nutzungsweise und Funktion. Einprägsam schildert Norman dies im Zusammenhang mit dem Design von Türen, welche sich uns häufig als abrupte Barriere in den Weg stellen wenn uns die Gestaltung des Türgriffs dazu verleitet sie zur falschen Seite hin öffnen zu wollen. In Abschnitt 2.3.3 wird Normans Modell des Dialoges zwischen Mensch und Maschine im Kontext der speziellen Anforderungen behinderter Nutzer bereits angesprochen.

Das Konzept der *Affordances* bezieht sich insbesondere darauf wie ein Artefakt seine Funktion dem Nutzer vermittelt, also auf den Dialog zwischen Mensch und Maschine. Das Konzept der *mental Modelle* hingegen bezieht sich dagegen in erster Linie auf die Frage wie das Wissen *beim Benutzer* repräsentiert ist.

Die Vorstellung der mentalen Modelle lässt sich auf die Arbeiten von Craik [Cra43] zurückführen. Dieser stellte fest, dass sich die menschliche Natur bei der Erklärung von Phänomenen nicht auf eine rationale Analyse der Gegebenheiten stützt, sondern vielmehr eine innere, unvollständige Abbildung der Realität pflegt, welche vielmehr als Basis für Erklärungen genutzt wird. Es scheint, so Craik, als ob der Mensch aus seiner Erfahrung, Wahrnehmung und Interpretation kleine Modelle der Realität bilde und diese als Erklärungsmodell und als Testfall zur Überprüfung von Hypothesen nutze.

“[With this representation in mind, people are] able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and future, and in every way to react in a much fuller, safer, and more competent manner to emergencies which face it”
[Cra43]

Größere Beachtung erhielt dieses Konzept im Bereich der Usability-Forschung erst 40 Jahre später, als in demselben Jahr zwei Bücher mit dem Titel *Mental Models* erschienen, welche beide das kognitive Phänomen der mentalen Modelle aus verschiedenen Blickwinkeln betrachteten:

- Die Arbeit von Johnson-Laird [JL83] ist in erster Linie kognitionspsychologisch motiviert und bezieht sich direkt auf die Thesen von Craik. In zehnjähriger Forschung hat Johnson-Laird ein Modell menschlicher logischer Schlussfolgerung entwickelt, welches wesentlich auf den Überlegungen zu mentalen Modellen basiert. Johnson-Lairds Ausgangspunkt ist die Beobachtung, dass Menschen bei der Lösung einer bestimmten Art logischer Probleme, sogenannten *Syllogismen*, offensichtlich durch die ökologische Validität der Aufgabe beeinflusst werden.
- Das zweite Werk, welches zeitgleich mit Johnson-Lairds Arbeit erschien, beinhaltet eine Sammlung von Artikel die von Gentner und Stevens [GS83] herausgegeben wurde. Diese betrachten das Phänomen von einer technischen und anwendungsorientierten Perspektive. Insbesondere dem ersten Beitrag von Norman [Nor83] gelingt es, wesentliche Eigenschaften mentaler Modelle aus einer anwendungsorientierten Sichtweise prägnant zusammenzufassen.

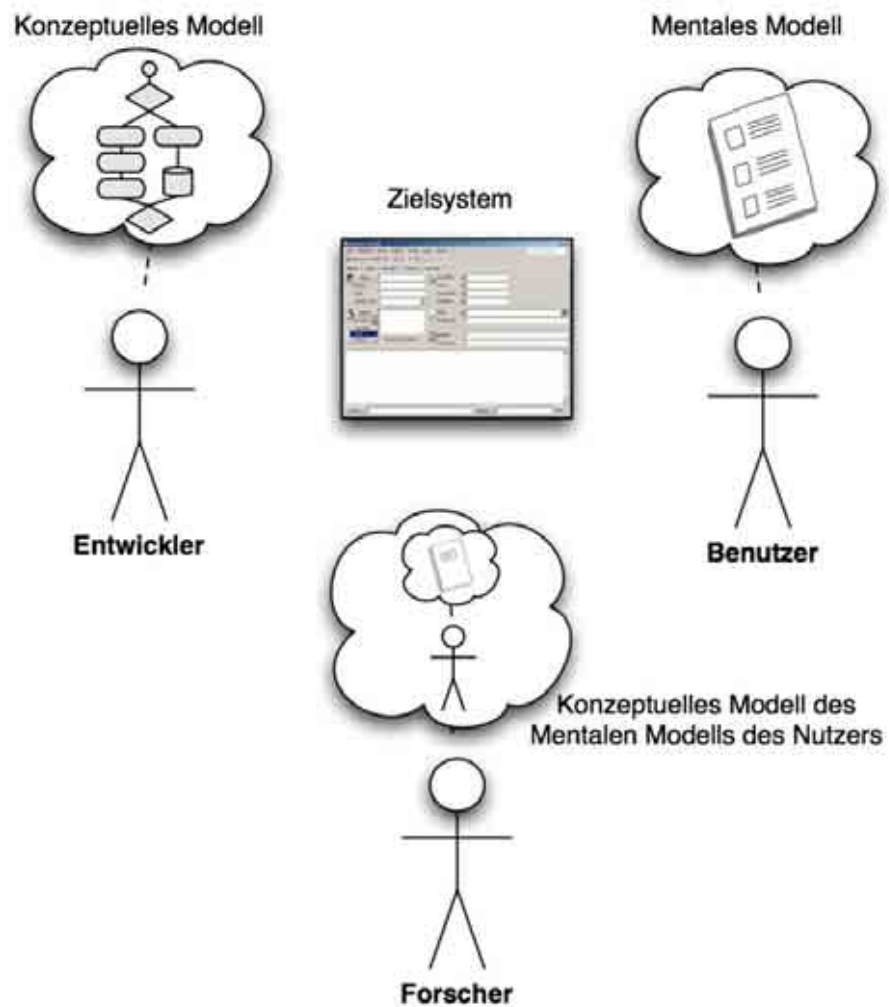


Abbildung 3.2: Übersicht der Modelle, welche bei der Erforschung des mentalen Modells eine Rolle spielen können.

Norman identifiziert vier Modelle, welche bei der Erforschung mentaler Modelle eine Rolle spielen können, und welche sich wechselseitig beeinflussen können (s. Abbildung 3.2).

Zielsystem: Das System, welches der Benutzer bedienen möchte.

Konzeptuelles Modell: Eine korrekte Repräsentation des Systems, wie sie zum Verständnis und Training eines Systems notwendig ist.

Mentales Modell: Eine reduzierte Repräsentation des Systems, so wie es der Benutzer wahrnimmt. Dieses entwickelt sich evolutionär während der Interaktion mit dem System und muss nicht technisch korrekt aber funktional sein.

Konzeptuelles Modell des Wissenschaftlers: Hypothese von dem mentalen Modell des Benutzers.

Die Repräsentation der mentalen Modelle ist implizit und repräsentiert visuelle aber auch abstrakte Inhalte. Es kann aus diesem Grunde vom Benutzer nicht oder nur schwer beschrieben werden. Auch sind die Konzepte in diesen Modelle nicht klar umrissen, sondern vielmehr in ein semantisches Netzwerk aus Konzepten eingebettet und somit für den Benutzer schwer zu fassen. Aufgrund ihrer schwer greifbaren Form gestaltet es sich als äußerst schwierig, mentale Modelle experimentell zu erfassen und zu erforschen.

Mentale Modelle sind vergänglich. Sie verändern sich mit den Erfahrungen des Benutzers, der seine interne Repräsentation stets mit der Realität abgleicht. Dies geschieht meist weniger in rationaler, logische Weise, als vielmehr bedürfnisgetrieben und häufig irrational, wenn die kausalen Zusammenhänge für den Benutzer nicht klar ersichtlich sind. Hat sich zum Beispiel eine Handlungsweise einmal als erfolgreich erwiesen, scheint diese gegenüber Training häufig recht immun. Der Benutzer behält Überzeugungen bei, die erprobt und funktional sind.

Dieser zielorientierte praktische Aspekt mentaler Modelle äußert sich laut Norman in einer gewissen *Faulheit*: mentale Modelle sind so aufgebaut, dass der kognitive Aufwand minimal gehalten wird. Dies kann dazu führen, dass Benutzer umständliche aber einfache Lösungen gegenüber kognitiv aufwändigeren aber schnelleren Lösungen bevorzugen. Scheint eine Methode sich in der Vergangenheit bewährt zu haben, wird der Benutzer versuchen, diese möglichst lange beizubehalten, selbst wenn diese Strategien vielmehr auf *Aberglauben* als auf tatsächlicher Funktion des Systems basieren. Norman [Nor83] beschreibt dies wie folgt.

“These doubts and superstitions govern behavior and enforce extra caution when performing operations. This is especially apt to be the case when a person has experience with a number of different systems, all very similar, but each with some slightly different set of operation principles.”

Wie sich noch in Abschnitt 3.3 zeigen wird, hat Norman hier ein grundlegende Eigenschaft der Interaktion zwischen Menschen und Computern identifiziert: *Menschen versuchen existierendes Wissen beizubehalten und können davon profitieren, wenn dies bei der Interaktion zu einem Vorteil wird. Unvorhersagbare Veränderungen oder Inkonsistenzen unterstützen den „Aberglauben“ des Benutzers und reduzieren das Vertrauen in vorhandenes Wissen.*

Weiterhin zeigt Norman mit dem Hinweis auf die Faulheit des Benutzers auf, dass dieser stets versuchen wird, kognitiven Aufwand dadurch zu reduzieren, dass er sein Repertoire an Handlungsregeln minimal hält und diese über Anwendungen zu generalisieren versucht. Benutzer werden aus diesem Grund versuchen allgemeine Regeln zu verwenden, die über verschiedenen Anwendungen hinweg genutzt werden können, anstatt für jede Anwendung neue Regeln bereitzuhalten.

Mentale Modelle spielen offensichtlich bei der Interaktion zwischen Mensch und Maschine eine wichtige Rolle. Welche Aspekte des mentalen Modells müssen bei deren Betrachtung berücksichtigt werden? Norman teilt ihre Funktion in drei wesentliche Faktoren ein.

Überzeugung (*Belief System*): Ein mentales Modell enthält die Überzeugungen des Benutzers, welche dieser über den Aufbau und die Funktionsweise einer Anwendung besitzt.

Beobachtbarkeit (*Observability*): ein mentales Modell repräsentiert die beobachtbaren Systemzustände und stellt somit ein internes Abbild des System und seiner Zustände dar.

Vorhersagekraft (*Predictive Power*): das mentale Modell ermöglicht es dem Benutzer, System-Reaktionen und Verläufe vorherzusagen. Dazu muss das Modell die zur Vorhersage relevanten Eigenschaften und Rand-Parameter berücksichtigen und repräsentieren.



Abbildung 3.3: Vergleich der Zahlenfelder auf einer Computer-Tastatur und auf einem Telefon. Die Inkonsistente Anordnung der Tasten kann leicht zu Fehlern führen.

3.3 Konsistenz

[...] consistency in computer systems constitutes a promise to the user. And it is not polite to break a promise.

— Jacob Nielsen, *Coordinating User Interfaces for Consistency*, 1989

Die konsistente Gestaltung von Produkten im Allgemeinen und Benutzerschnittstellen im Besonderen wird als eines der wichtigsten Kriterien benutzbarer Systeme erachtet: “one of the most important aspects of usability is consistency in user interfaces” [Nor88]. Auch Tognazzini nahm die Konsistenz in seine wichtigsten Prinzipien des Interaktionsdesigns auf [Tog03, Tog90].

Auch die Design-Richtlinien der wichtigsten Hersteller von Betriebssystemen, die *Microsoft Windows Interface Guidelines for Software Design* [Mic98] sowie die *Apple Human Interface Guidelines* [App92] stellen die Bedeutung von Konsistenz in den Vordergrund. So heißt es in *Microsofts* Richtlinie, „consistency is important through all aspects of the interface, including names of commands, visual presentation of information, and operational behavior“. Apple zügelt den Gestaltungswillen des Designers mit den Worten, „in most cases, consistency should be valued above idiosyncratic cleverness“.

Ben Shneiderman stellt Konsistenz an den Anfang seiner acht Goldenen Regeln der User Interface Design und überschreibt diese plakativ mit den Worten, „Strive for consistency“ [Shn98].

Konsistenz ist zugleich ein intuitives und schwer greifbares Konstrukt, über dessen Wirkungsweise und Einflussfaktoren wenig Einigkeit herrscht [Gru92]. Die konsistente Umsetzung von Geräten und Anwendungen gestaltet sich in der Praxis als vielschichtig und komplex und, so können wir dies häufig in unserer alltäglichen Umgebung betrachten, scheitert nicht selten auf fatale Weise (s. Abbildung 3.3).

Die uneinheitliche Sicht auf die Rolle der Konsistenz im Design von Benutzerschnittstellen zieht sich durch die gesamte Literatur und die dort verfügbaren Definitionen. Ben Shneiderman [Shn98, S. 74] definiert Konsistenz anhand seiner Designkriterien, welche für die konsistente Gestaltung eines User Interfaces notwendig sind: durch ähnliche Handlungsfolgen in ähnlichen Kontexten, einheitliche Terminologie, durchgängige Farbgebung, Layout und Typographie. Im Falle von Abweichungen sollte dies dem Benutzer deutlich gemacht werden (Transparenz, s. auch [DK04]). Jakob Nielsen [Nie93] weist auf die Bedeutung der Platzierung und Formatierung von Elementen in einer graphischen Oberfläche hin und empfiehlt, diese konstant zu halten.

Empirische Belege für die Vorteile konsistenter User Interfaces existieren in großer Zahl. So zeigen u. a. Payne und Green, dass konsistente Benutzerschnittstellen leichter erlernbar sind [PG86, PME86, MS97]. Andere Untersuchungen zeigen, dass Konsistenz dazu beiträgt die Häufigkeit von Fehlern bei der Bedienung zu reduzieren [PG89, Doy90] und die Bedienbarkeit zu verbessern [Tul88a]. Graphische Designern ist Konsistenz als gestalterisches Mittel zur Ordnung von Elementen in Übereinstimmung mit deren inneren Struktur bekannt [VG94, Le 53].

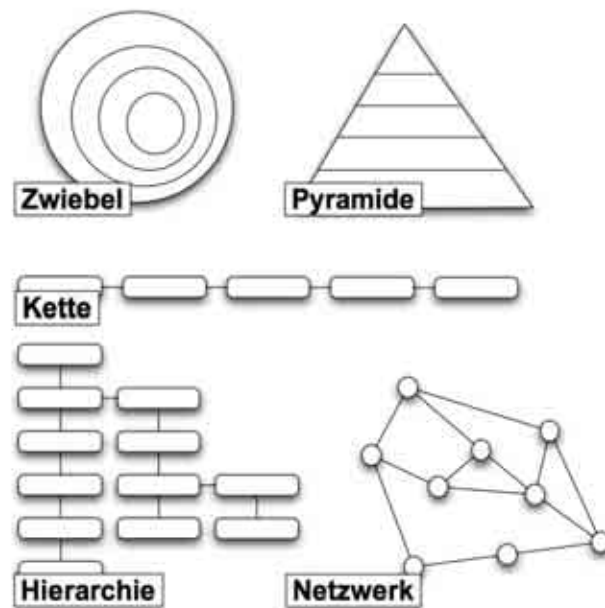


Abbildung 3.4: Alternative Mentale Repräsentationen der Menüstruktur nach Ziefle *et al.*[ZAB06].

Dass Konsistenz nicht nur die graphische Darstellung betrifft, sondern auch die Gestaltung der Aufgabenstruktur, der Interaktionskonzepte und der Rückmeldungen an den Benutzer zu verbessern in der Lage ist, haben verschiedene Untersuchungen gezeigt. Eine Studie von Kellogg [Kel87] zeigt, dass eine visuell konsistente, jedoch konzeptuell inkonsistente Variante einer Oberfläche langsamer bedienbar sind und die Benutzer weniger zufrieden stellen. Eberts und MacMillan [EM87] zeigen, dass der Wechsel zwischen zwei Kommandozeilen-Editoren, die zwar die gleiche Darstellung und dieselben Interaktionskonzepte umsetzen, jedoch auf völlig verschiedenen Systemen laufen, zu Fehlern aufgrund erheblicher Konsistenzprobleme führen.

Weitere empirische Unterstützung erhalten die Befürworter konsistenter Gestaltung aus der Forschung zu kognitiven Architekturen. Produktionssysteme wie *Soar* [New90, LNR87] und *ACT-R* [And93, SA89] haben sich als geeignet erwiesen, um Konsistenzeffekte beim Wissenstransfer zwischen Anwendungen vorherzusagen. Polson *et al.*[PME86] konnten zeigen, dass Unterschiede in der Reaktionszeit beim Erlernen verschiedener Anwendungen aufgrund der Anzahl der neu zu lernenden Regeln vorhergesagt werden kann. Auch Singley und Anderson konnten den Transfer von Wissen zwischen Texteditoren nachweisen [SA89].

Ziefle *et al.*[ZAB06, ZB04] untersuchten die Auswirkung der mentalen Repräsentation von Menüstrukturen auf die Interaktion mit mobilen Endgeräten (Mobiltelefon, PDA). Sie stellen fest, dass die Verfügbarkeit der korrekten mentalen Repräsentation der Menüstruktur den Umgang mit der Anwendung deutlich erleichtert (s. Abbildung 3.4). Ziefle *et al.* verglichen die Unterschiede zwischen jüngeren und Älteren Benutzern und stellten fest, dass insbesondere die Jüngeren von der Konsistenz zwischen tatsächlicher hierarchischer Menüstruktur und ihrer Vorkenntnis aus anderen Programmen profitieren konnten, während die Älteren häufig inkonsistente mentale Repräsentationen aufwiesen und dadurch schlechtere Leistungen im Umgang mit der Anwendung zeigten.

Kellogg [Kel89] fasst das Wesen der Konsistenz treffend zusammen mit dem Satz:

“Issues of consistency, both within and across applications and system environments, are inescapable for designers and users: users generalize, correctly or incorrectly, on the basis of their previous experience and what they know, and applications and environments are consistent or inconsistent in a variety of ways that impact use.” [Kel89]

Tognazzini [Tog03, Tog90] stellt ebenfalls die Beziehung der Anwendungslogik mit der Erwartung des Benutzers in den Vordergrund indem er sagt “The most important consistency is consistency with

user expectations” [Tog03]. Daher kann Konsistenz gezielt eingesetzt werden, wenn unterschiedliche Verhalten der Anwendung signalisiert werden sollen.

Konsistenz ist somit ein Konzept, welches die *Beziehung* zwischen einem Artefakt und der Erwartung, bzw., der mentalen Repräsentation eines Menschen von diesem Artefakt, darstellt. Reisner [Rei90] stellt fest, dass Konsistenz keine feste Eigenschaft eines Systems oder eines Nutzers ist, sondern die Beziehung zwischen zwei Sprachen: der Sprache des Designers, welcher durch sein Design kommuniziert, und der Sprache des Nutzers, der diese Nachricht empfängt [DS05].

Konsistenz ist kein universelles Maß, sondern stets abhängig davon, in Bezug auf welches Kriterium die Konsistenz bewertet werden soll [Kel89]. Ein Design kann so zum Beispiel bezüglich der Positionierung konsistent und gleichzeitig von völlig inkonsistenter Farbgebung sein. In einer Situation sind aus diesem Grunde auch nur jene Dimensionen der Konsistenz, welche die Aufgabe oder den Verwendungszweck des betreffenden Artefakts beeinflussen oder welche mit diesen in Wechselwirkung stehen [BHB04].

Betrachten wir die Natur der menschlichen Informationsverarbeitung (s. Abschnitt 3.2), stellen wir fest, dass der Austausch zwischen Designer und Benutzer als Wirkungsebene konsistenter Gestaltung einen Austausch von Informationen, also Wissenstransfer, zwischen Menschen darstellt. De Souza [DS05] diskutiert diesen Prozess im Lichte der Methoden der *Semiotik* bietet. De Souza interpretiert eine Benutzerschnittstelle im semiotischen Sinne als Symbolsystem, durch welches die Nachricht des Designers vermittelt wird.

Nach Charles Peirce [Pei58] einem der Vorreiter der Semiotik, übermittelt ein Symbol Information, indem ein Repräsentant (*representamen*) genutzt wird um auf eine Bedeutung (*interpretant*) und evtl. auf ein zugrunde liegendes Objekt (*referent*) zu verweisen. Erst die Interpretation des Symbols durch den Betrachter macht es zu einem Symbol, indem dieser eine Bedeutung in den Repräsentanten interpretiert und dieses so mit Bedeutung füllt. Die Interpretation des Symbols erfolgt nach Peirce in Interpretationszyklen (*reasoning cycles*), die entweder *deduktiv* aus bekannten Regeln Ergebnisse ableiten, oder *induktiv* aus beobachteten Kontingenzen Regeln abzuleiten versuchen.

Norman spricht hierbei vom Menschen als *explanatory creatures* [Nor88]. Im Sinne von Jakobsons Kommunikationsmodell [Jak60] stellt das Design also die Nachricht dar, welche der Designer dem Benutzer übermittelt. Die gestalterischen Mittel wiederum stellen den Code dar, in welchem die Nachricht ausgedrückt wird. Der Code, sprich die Symbolik, entsteht über die Interpretation des Betrachters in jedem Falle neu, da diese vom Vorwissen, also dem mentalen Modell des Betrachters abhängt.

Eine konsistente Umsetzung der Darstellung, der Interaktion und der inneren Abläufe eines Systems hilft dem Benutzer dabei, sein mentales Modell durchgängig zu nutzen. Je konsistenter die Umsetzung, desto weniger neue Regeln müssen induziert und in das bestehende mentale Modell integriert werden, desto zuverlässiger kann das vorhandene Wissen dann auch auf neue Situationen angewandt, also deduziert werden [Nie89].

“Consistency is assumed to enhance the user’s possibility for transfer of skill from one system to another. By doing so, consistency leads to ease of learning and ease of use.”
[Nie89, S. 4]

Norman [Nor83] stellt fest, dass Menschen häufig bereit sind mehr Aufwand in Kauf zu nehmen, wenn sie dadurch mentale Anstrengung vermeiden können. Inkonsistentes Verhalten erfordert Planung und Beachtung jedes Einzelfalles, wohingegen Konsistenz die Automatisierung unterstützt [Wan03b]. Eben diese Tatsache, der Konflikt zwischen Prozessoptimierung und Konsistenz, motivierte Grudin [Gru92, Gru89] zu seiner mittlerweile berühmten Replik auf das *ITS* System von Wiecha *et al.* [WBBG90]. Grudin warnt vor Übersimplifizierung. Konsistenz, so Grudin, ist kein Allheilmittel und kann in manchen Fällen sogar kontraproduktiv sein: „Consistency is a trade-off against other goals: at times, it is not the best design strategy.” Grudin belegt dies mit einem Beispiel zur Menü-Auswahl bei der konsistente Positionierung der initialen Selektion die aufgabenbezogene Ausführung behindern kann.

Konsistenz muss also gegen andere Kriterien abgewogen und unter Umständen zugunsten dieser zurückgestellt werden. Dieser Konflikt kann in der Regel nur durch den Designer aufgelöst wer-

den. Diesem die hierfür notwendige Unterstützung zu bieten ist ein zentraler Aspekt der hier vorgestellten Methoden zur Entwicklung plattformübergreifender Benutzerschnittstellen und wird in den Abschnitten 2.4, 5 und 6.2 ausführlich dargestellt.

3.3.1 Dimensionen der Konsistenz

Wie die Betrachtungen im vorherigen Abschnitt gezeigt haben, steht die Erkenntnis von der Bedeutung der Konsistenz bei der Gestaltung von Benutzerschnittstellen einer relativen Ratlosigkeit bei deren Messung. Reisner [Rei81] drückt diesen Widerspruch aus mit den Worten „It is almost a truism that consistency of an interface will make it easier to use. What is less clear, however, is precisely what we mean by consistency and, more importantly, how do we identify its absence.“

Den vorherigen Abschnitt zusammenfassend, kann festgestellt werden dass Konsistenz keine feste Eigenschaft eines Dialoges ist, sondern die Beziehung zwischen verschiedenen Eigenschaften eines Dialoges, zwischen der Eigenschaften des Dialoges und der Erwartung des Benutzers, sowie zwischen den Eigenschaften des Dialoges und dem realen Sachverhalt, der über diesen Dialog abgebildet wird. Diese Perspektiven sind naturgemäß nicht unabhängig, sondern bedingen einander. Weiterhin sind die Ausprägungen dieser Eigenschaften nicht statisch sondern können sich dynamisch, in Abhängigkeit vom Betrachter, dessen Kontext, sowie des Kontextes des Dialoges, verändern. Verdeutlichen wir uns die Wirkungsweise der Konsistenz auf den menschlichen Betrachter, zeigt sich, dass die mentale Repräsentation, das mentale Modell, ebenfalls diese Eigenschaften: Veränderlichkeit, Abhängigkeit von Kontext und Unvollständigkeit besitzt.

Konsistenz ist letztendlich ein Effekt, der die Übereinstimmung des Dialoges mit dieser mentalen Repräsentation ausdrückt.

Tognazzini [Tog03] definiert sieben Ebenen auf denen Konsistenz im Interaktionsdesign von besonderer Bedeutung ist. Die folgende Auflistung zeigt diese Ebenen in absteigender Bedeutung.

1. Interpretation des Benutzerverhaltens (z.B. identische Tastaturkürzel).
2. Unsichtbare Strukturen (z.B. Funktionen die bekannt sein müssen).
3. Kleine sichtbare Strukturen (wie Funktionen verschiedener Rahmentypen).
4. Darstellung und Design einer Anwendung.
5. Darstellung und Design einer Produktfamilie.
6. Konsistenz der Darstellung innerhalb einer Organisation / eines Herstellers.
7. Konsistenz innerhalb einer Plattform.

Dieses Ordnungsschema, welches insbesondere durch seinen heuristischen Wert besticht, wird im Referenzmodell der plattformübergreifenden Konsistenz (s. Abschnitt 5.4.2) in einen generellen Bezug gesetzt werden.

Stellt man Konsistenz als relationales Phänomen dar, ist es notwendig, die Dimensionen der Beziehungen zu erfassen und diese zu analysieren. Kellogg unterscheidet hierbei zum einen die inhaltlichen Dimensionen der Konsistenz, d. h. bezüglich *welches Kriteriums* ist eine Anwendung konsistent, und eine referenzielle Dimension, d. h. *welche Systeme* werden verglichen [Kel87].

Die *inhaltlichen Dimensionen* orientieren sich an den konzeptuellen Ebenen der Interaktion wie sie von Moran [Mor81] oder in ähnlicher Form von Anderen [Rei81, PG86, Nor88, FD82] definiert wurden. Ein interaktives System lässt sich nach Moran in drei Schichten gliedern, welche jeweils zwei Ebenen enthalten.

Konzeptuelle Schicht: Sie beinhaltet jene Ebenen der Interaktion, welche die Umsetzung der internen, werkzeugunabhängigen Repräsentation der Aufgabe, d. h. des Handlungs- oder prozeduralen Wissens des Benutzers in die Konzepte des Werkzeugs, bzw. der Software repräsentieren.

Aufgaben-Ebene: Sie repräsentiert die Handlungsziele. Auf dieser Ebene werden die werkzeugunabhängigen Methoden und kognitiven Fertigkeiten zur Analyse und Zerlegung der Gesamtaufgabe in Einzelschritte repräsentiert.

Semantische Ebene: Sie repräsentiert die Übersetzung der werkzeugunabhängigen Methoden der Aufgaben-Ebene auf werkzeugabhängige Funktionen. Die Beschreibung der Funktionen im User Interface des Systems wird also im Sinne der Nutzbarkeit für die Aufgabenerfüllung bewertet.

Kommunikationsschicht: Sie beinhaltet jene Ebenen der Interaktion, auf denen der Austausch zwischen System und Benutzer stattfindet. Der Benutzer muss die Sprache des Systems verwenden, um die geplanten Handlungssequenzen zu kommunizieren.

Syntaktische Ebene: Auf ihr wird die Aktivierung der auf der semantischen Ebene ausgewählten Funktion erfolgen. Die Interaktionsmöglichkeiten, welche das System zur Verfügung stellt, sind hier repräsentiert.

Interaktionsebene: Hier wird die Handlungsfolge in eine Sequenz von konkreten Eingabeoperationen, wie Mausinteraktion oder Tastendruck übersetzt.

Physikalische Schicht: Hier befinden sich die konkreten Realisierungen des Systems und der Interaktionsmittel.

Ebene der räumlichen Gestaltung: Sie repräsentiert die wahrnehmbaren Gestaltungsmerkmale des Systems, wie das Layout eines Graphischen User Interfaces aber auch das der einzelnen Interaktionselemente.

Geräte-Ebene: Sie berührt die physikalische Umsetzung der Darstellung und Interaktion im Sinne der verwendeten Hardware.

Die *referenzielle Dimension* unterscheidet hingegen zwischen der Konsistenz innerhalb eines Systems (*intern*) und bezüglich eines Außenkriteriums (*extern*), wie beispielsweise der Erwartung des Benutzers, allgemeiner Konventionen oder physikalischer Gegebenheiten. Tabelle 3.1 zeigt ein Klassifikationsschema zur Definition der Dimensionen der Konsistenz, welches von Kellogg vorgeschlagen wurde [Kel87, Kel89].

		Intern	Extern
Konzeptuell	Aufgaben-Ebene	Zerlegung in analoge Aufgaben	Vergleich Aufgabenmodell Nutzer vs. Anwendung
	Semantische Ebene	Semantische Abläufe, Vollständigkeit	Zusammenhang mit Welt oder Metapher
Kommunikation	Syntaktische Ebene	Organisationsprinzipien, Ähnlichkeit	
	Interaktionsebene	Organisatorische Prinzipien der Kommandos, Position der Argumente	Kongruenz der Kommandos mit natürlicher Sprache
Physikalisch	Räumliches Layout	Räumliche Anordnung, visuelle Charakteristika	
	Geräte-Ebene		

Tabelle 3.1: Kelloggs Schema zur Definition von Konsistenz.

Kellogg bietet somit die Möglichkeit, die verschiedenen Aspekte der Konsistenz einzuordnen. Die Merkmale eines User Interfaces können einer konzeptuellen Ebene zugeordnet werden. Die Konsistenz dieses Merkmales innerhalb des Systems, bzw. zwischen System und einem Außenkriterium kann erfasst und bewertet werden. Die Konsistenz der gesamten Anwendung kann so als Kombination der Einzelkonsistenzen beurteilt werden. Dennoch bietet Kellogg mit diesem Framework lediglich einen Rahmen und nicht die Methoden zur Erfassung der Konsistenz, bzw. der Dimensionen des Systems.

Kelloggs Modell bildet die verschiedenen Aspekte der Konsistenz direkt auf die Ebenen der gängigen Modellen der Mensch-Maschine Kommunikation [Nor88, FD82] ab. Damit ist es jedoch nicht in der Lage, die Interaktion zwischen den Dimensionen zu erfassen. Aspekte des Kontextes werden nur

ansatzweise erfasst oder fallen aus dem Modell. Somit ist dieses Modell nur bedingt geeignet die Dimensionen der Konsistenz in plattformübergreifenden multiplen Benutzerschnittstellen zu beschreiben. In Abschnitt 5.4 wird ein Referenzmodell der plattformübergreifenden Konsistenz eingeführt werden.

Wie verschiedene Untersuchungen zeigen, besteht auch bei ausgebildeten Designern keine Einigkeit darüber, wie genau Konsistenz zu messen ist. In Abschnitt 4.4 werden konkrete Aspekte der User Interface Gestaltung aus der Literatur gesammelt und den Dimensionen aus Kelloggs Framework zugeordnet.

4 Methoden der Entwicklung von Benutzerschnittstellen

Moderne PC Anwendungen bieten zunehmend komplexe Interaktionsmöglichkeiten und eine qualitativ hochwertige Graphik. Im gleichen Maße wie die Komplexität dieser Anwendungen gestiegen ist, sind die Anforderungen an deren Benutzbarkeit und somit die Anforderung an die Konsistenz, die Geschwindigkeit, die selbsterklärende Gestaltung, die Organisation, die Fehlertoleranz und Korrigierbarkeit, die Übersichtlichkeit und Einfachheit der Anwendungen gewachsen [Shn98]. Um die Entwicklung solcher Anwendungen zu erleichtern und zu beschleunigen, wurden Werkzeuge zur Entwicklung von Benutzerschnittstellen entwickelt, die sich nach Foley und Van Dam [FD82] in vier Abstraktionsebenen gliedern.

Abbildung 4.1 stellt diese Abstraktionsebenen dar. Diese setzen auf dem Betriebssystem auf und gliedern sich in *Window-System*, die *User-Interface Toolkits* und die *User-Interface Management Systeme* (UIMS) [DFB92, Göt95].

4.1 Windows-System

Ein *Windows-System* [BB91] dient als Hardware-unabhängige Abstraktion für den Entwickler von graphisch-interaktiven Systemen.

Das *Basis-Windows-System* bietet eine abstrakte Schnittstelle zu den vorhandenen Zeige-Geräten, Tastaturen, Graphikkarten und Monitoren, welche ihrerseits über spezifische Gerätetreiber angesprochen werden. Die Beschreibung des abstrakten Terminals erfolgt über das *imaging model* [DFAB93]. Die Beschreibung von Bildpunkten als *Pixels*, die Kapselung von elementaren Graphik-Operationen in Makros wie im *Graphischen Kernel System (GKS)* [Enc80, ISO94] oder die Verwendung vektorbasierter Graphik-API, wie *Adobe PostScript* im *NextStep* System, oder das *Portable Document Format (PDF)* bei Apples *Mac OS X Quartz*.

Der *Window-Manager*, als Teil des Windows-Systems, verwaltet die Darstellung verschiedener Anwendungen, welche auf die Funktionen des Basis-Windows-Systems zugreifen. Die Positionierung, Aktivierung und Layout von Anwendungen in Fenstern können über den Window-Manager interaktiv gesteuert werden.

Betrachtet man die Möglichkeiten der plattformübergreifenden Entwicklung wird schnell klar, dass die Abstraktionsebene eines Windows-Systems kaum ausreichend ist, um die notwendigen Anpassungen an Gerät, Anwendungs- und Nutzungskontext zu ermöglichen, die hier notwendig sind.

4.2 User Interface Toolkits

Eine zentrale Eigenschaft von WIMP-basierten Benutzerschnittstellen ist die Tatsache, dass der Benutzer Ein- und Ausgabe mit klar unterscheidbaren unabhängigen Elementen, den *Interaktoren* (auch als Widgets, Controls oder Bedienelemente bezeichnet), verknüpft. So entsteht der Eindruck, dass genau dies die Objekte sind, welche eigentlich von Interesse sind, und, dass diese direkt manipuliert werden müssen [DFAB93, Shn98]. Der User Interface Toolkit [WBBG90] stellt genau diese Elemente zur Verfügung. Hierzu setzt er auf die graphischen Ausgabemethoden und die Eingabe-Queue aus dem Window-Manager auf und sorgt dafür, dass sich diese Element auf Benutzereingaben erwartungskonform *verhalten*. Eine Schaltfläche *versinkt* in der Oberfläche wenn man auf sie klickt, und die Elemente in einer Liste werden *verschoben*, wenn man die Schiebeleiste bedient.

Ein wesentlicher Vorteil von User Interface Toolkits besteht darin, dass Interaktionselemente automatisch einheitlich dargestellt werden. Das so genannte *Look & Feel* einer graphischen Benutzerschnittstelle definiert sich durch genau diese Einheitlichkeit der Gestaltung und des Verhaltens. Auf

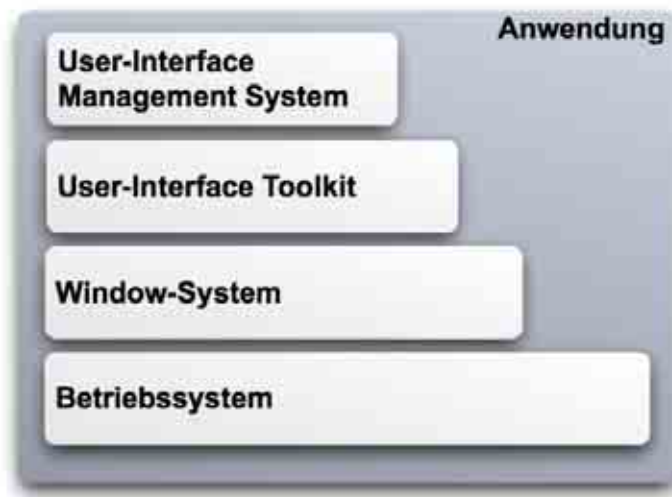


Abbildung 4.1: Abstraktionsebenen der Methoden zur Entwicklung von Benutzerschnittstellen

diese Weise entstehen anwendungs- oder betriebssystemweite einheitliche Darstellungen von Anwendungen. Eine gewisse Durchgängigkeit ist somit gewährleistet. Die Forderung nach Konsistenz erhält durch die Verwendung von User Interface Toolkits erhebliche Unterstützung.

User Interface Toolkits werden in den heute üblichen objektorientierten Programmiersprachen meist als Objekthierarchien realisiert, welche eine Menge an grundlegenden Elementen definieren. Es ist dem Entwickler nun möglich, eigene Elemente auf Basis dieser Toolkit-Elemente zu definieren indem er diese ableitet. Weiterhin ist die Möglichkeit gegeben, komplexe Eingabeelemente aus mehreren primitiven Eingabeelementen zu definieren und auch dynamisch zu verändern (so besteht eine Liste meist aus einer Fläche mit einer Schiebeleiste und mehreren Teilflächen, welche für die Darstellung der Listenelemente genutzt werden).

Auch die Behandlung von Benutzereingaben wird von Toolkits meist über eine *Ereignis*-Schnittstelle behandelt. Benutzereingaben werden meist in verschiedene Ereignisse abstrahiert, welche dann von einem Programm abgefangen und behandelt werden können. In den meisten modernen Sprachen steht hierfür eine Variante des *Delegaten*-Musters zur Verfügung.

Präsentationsvorlagen und Look & Feel In toolkitbasierten Ansätzen wird die anwendungsübergreifend einheitliche Darstellung häufig über Präsentationsvorlagen (engl. *Templates*) oder das so genannte *Look & Feel* umgesetzt. Templates definieren Darstellungseigenschaften in einer gesonderten Ressource, die getrennt von der eigentlichen Spezifikation der Benutzerschnittstelle gehalten wird. Einheitliche Darstellungsoptionen, wie z.B. Farbsätze, Schriftarten, aber auch die Dekoration und das Verhalten von Elementen, können so beschrieben und für verschiedene Anwendungen eingesetzt werden. Ein Beispiel für eine solche Beschreibungssprache sind die *Cascading Style Sheets (CSS)* [LB99a]. Abbildung 4.2 zeigt die GUIDE Entwicklungsumgebung für HMI-Systeme im Automobilbau. Hier können Templates für die konsistente Gestaltung der Benutzerschnittstelle des Bordcomputers eingesetzt werden.

Ein ähnlicher Ansatz existiert mit den sogenannten *Skins*, die ebenfalls als getrennte Ressourcen wie eine Haut auf die Oberfläche gelegt werden können. Hier beschränkt sich die Anwendbarkeit allerdings primär auf graphische Aspekte. So genannte *Look & Feel* werden z. B. bei den Java-Swing Toolkits angeboten. Diese stellen gesonderte Objekte dar, die im Prinzip der *Model-View-Controller*-Architektur lediglich die Darstellung der Elemente regeln. Diese werden bei der Darstellung vom Toolkit verwendet und können dynamisch zur Laufzeit gewechselt werden. So können die plattformunabhängigen Implementierungen plattformspezifische Darstellungen erhalten.

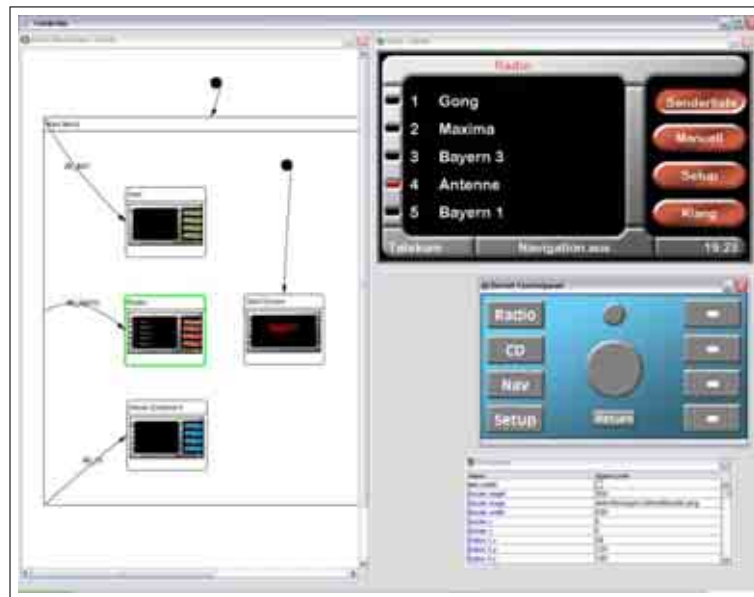


Abbildung 4.2: GUIDE Entwicklungsumgebung der Firma 3Soft.

Ein wesentliches Problem der *Templates*, *Skins* bzw. *Look & Feels* ist die Tatsache, dass sich diese auf die Darstellung beschränken. Eine Umstrukturierung bzw. umfassendere Anpassung der Oberfläche ist hiermit nicht oder nur sehr eingeschränkt möglich. Dennoch ist die Abtrennung gestalterischer Eigenschaften von der Struktur im Sinne der Modularisierung der Oberflächendarstellung ein wichtiger Aspekt und soll auch im hier vorgestellten Ansatz verfolgt werden.

Eine Anzahl von plattformübergreifenden User Interface Toolkits existieren auf dem Markt. Plattformübergreifende User Interface Toolkits stellen in der Regel plattformspezifische Bibliotheken zur Verfügung, welche über identische, oder zumindest ähnliche Schnittstellen anzusprechen sind und somit in Anwendungen eingebunden werden können. Dieses Konzept eignet sich insbesondere in Kombination mit plattformübergreifenden Programmiersprachen, wie *Sun Microsystems Java* [Kru04b] oder der *Microsoft .Net Plattform* [Pro02]. Die Toolkit-Bibliotheken werden in solchen Fällen meist dynamisch zur Laufzeit in die Anwendung eingebunden. Sowohl die Java-Plattform (Java AWT und Swing) als auch .Net (Windows Forms und Web Forms) stellen eigene Toolkit-Bibliotheken zur Verfügung, welche dann auf verschiedenen Endgeräten eingesetzt werden können.

Einige Forschungsansätze verfolgen ebenfalls die Möglichkeit der multimodalen Darstellung von User Interfaces auf Toolkit-Ebene wie *FlexClock* [GVRV02]. *FlexClock* ist eine Uhrzeit und Kalender Element welches dem *Plastic User Interfaces* [DCCD03] Konzept folgt und sich dynamisch an verschiedene Darstellungsgrößen anpasst. Darstellungseigenschaften wie die Art der Uhr (digital, analog), Ansicht des Datums (Kalender, Zahlen) wechseln zur Laufzeit und füllen den verfügbaren Platz optimal aus. Ein anderer Ansatz ist der *Multimodal Toolkit* von Crease *et al.* [CGB00], der durch eine Abstraktionsschicht die konkreten Darstellung unabhängig beschrieben wird und so eine Anpassung auf verschiedene Modalitäten (Sprache und Graphische UI) zur Laufzeit erreicht. *HOMER* [SS95] ist ein weiteres multimodales Toolkit System, welches ebenfalls die Anpassung an akustische und graphische UI ermöglicht. Solche Toolkits bilden bereits den Übergang zu den UIMS, die auf der Ebene der Dialogstruktur abstrahieren.

Die Umsetzung von plattformübergreifenden User Interface Toolkits ist bereits recht fortgeschritten, so dass seit einigen Jahren verschiedene Lösungen als Produkte auf dem Markt sind. Die folgenden existierenden Toolkits werden in der Folge kurz diskutiert.

- Sun Microsystems Java Swing / AWT
- Microsoft .Net WinForms / WebForms
- TrollTech QT
- Smalltalk

Sun Microsystems Java Swing / AWT Der *Advanced Windowing Toolkit (AWT)* und die Klassen der *Swing* Bibliothek stellen die Oberflächenbibliotheken für die *Java Plattform* zur Verfügung. Swing ist eine Erweiterung von AWT und die heute gebräuchliche, leistungsfähigere Lösung. Diese Toolkits bieten hierbei außer einer möglichst originalgetreuen Darstellung auf verschiedenen Plattform keine weiteren Anpassungen an die Plattform an. Ein plattformübergreifender *Look & Feel* kann definiert werden um so die Darstellung zu optimieren, wird jedoch nicht angeboten. Durch das Konzept der Layout-Manager kann zwar eine begrenzte dynamische Anpassung an die Darstellungseigenschaften einer Plattform vollzogen werden, allerdings stehen auch hierfür keine vorgefertigten Lösungen bereit, sondern müssten selbst realisiert werden. Weiterhin ist die Bandbreite einer Anpassung durch eine Layout-Management-Komponente eingeschränkt.

Der Aufbau der Java Plattform ist in verschiedene skalierbare Plattformen unterteilt, wobei die für Desktop PCs konzipierte *Java Standard Edition (J2SE)* den umfangreichsten Funktionsumfang bietet. Insbesondere für mobile Geräte stellt *Sun Microsystems* mit der *Java Microedition (J2ME)* eine Plattform zur Verfügung, welche auf die Leistungsmerkmale, wie auch auf die Benutzungsszenarien mobiler Anwendungen angepasst ist. Leider sind sowohl der Umfang, als auch die Schnittstellspezifikationen derart unterschiedlich, dass eine plattformübergreifende von Anwendung, d. h. der Wechsel zwischen *J2SE* zu *J2ME* und umgekehrt stets mit einer Portierung verbunden ist und nicht automatisch erfolgen kann. Die *Java Personal Edition (Personal Profile)* bzw. *Connected Limited Device Configuration (CLDC)* stellt eine reduzierte Version der *J2SE* dar, welche auch auf mobilen Geräten einsetzbar ist. Zur Implementierung der User Interfaces steht hier lediglich ein Kern an AWT Elementen zur Verfügung. plattformspezifische Anpassungen der Darstellung werden von diesem Toolkit nicht zur realisiert.

Microsoft .Net Plattform Als weiterer plattformeigene User Interface Toolkit stellt *Microsoft* in der .Net-Plattform die *Windows Forms* bzw. *Web Forms* zur Verfügung. Während die *Web Forms* für Client-Server basierte Web-Anwendungen konzipiert wurde (eine ähnliche Möglichkeit existiert mit den *JavaServer Faces* [Bos04] auch für die Java Plattform), treten die *Windows Forms* die Nachfolge der *Microsoft Foundation Classes (MFC)* auf der .Net Plattform an.

Ebenso wie die Java Plattform wird bei der .Net Plattform die Abkopplung des Programms von der Plattform durch einer Laufzeitumgebung, der sog. *Virtuellen Maschine* oder in der .Net Terminologie *Common Language Runtime (CLR)* realisiert. Diese interpretiert den vorkompilierten, plattformunabhängigen *Byte-* oder *Intermediate Code* erst zur Laufzeit in plattformspezifische Anweisungen.

Die Laufzeitumgebung stellt also die Anpassung auf die Zielplattform bereit, indem sie die notwendigen Anweisungen für den Zwischencode plattformspezifisch umsetzt. Durch die Verfügbarkeit einer CLR sowohl für die Desktop PC als auch für Mobilgeräte wie Pocket PCs und Smartphones (das .Net *Compact Framework*), steht hier dem Entwickler eine durchgehende Entwicklungsplattform bereit. Auch hier ist der Funktionsumfang, wie bei der Java Personal Edition im Vergleich zur Vollversion eingeschränkt. Anders als die *Java Personal Edition* stellt das .Net *Compact Framework* allerdings eine umfangreichere Anpassung der Benutzungsoberfläche an die Eigenschaften der Zielplattform dar. So werden Menüs und Symbolleisten auf die Eigenschaften der Pocket PC Plattform angepasst, indem sie, wie dort üblich am unteren Rand anstatt, wie auf dem PC am oberen Rand des Bildschirms dargestellt werden.

Durch die Integration beider Geräteplattformen in die Entwicklungsumgebung *Visual Studio* wird der Entwickler bei der Gestaltung von Benutzungsoberflächen in großem Maße unterstützt. Wie die Abbildungen 4.3 und 4.4 zeigen, können auf diese Weise Anwendungen für verschiedene Endgeräte in derselben Entwicklungsumgebung entwickelt werden. Allerdings muss hier die Zielplattform projektweit festgelegt werden so dass beide Plattformen nicht aus demselben Projekt bedient werden können. Die Nutzung gemeinsamer Bibliotheken ist bei Einhaltung einer gemeinsamen API möglich.

Trolltech QT *Qt* von *Trolltech* ist ein plattformunabhängiger Toolkit, welcher in C++ Anwendungen für Microsoft Windows, Mac OS und X11 auf Unix, sowie einige mobile Plattformen eingebunden werden kann. Über eine Abstraktionsschicht werden entweder native Oberflächenkomponenten

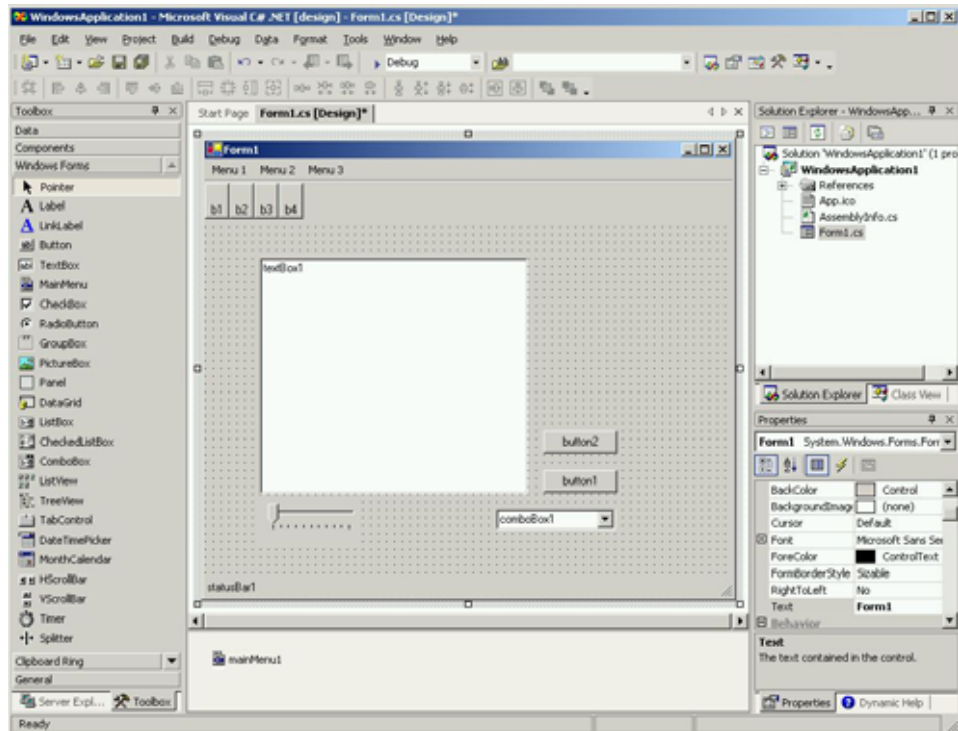


Abbildung 4.3: Der Graphische User Interface Designer der Entwicklungsumgebung Visual Studio mit einer Desktop Oberfläche.

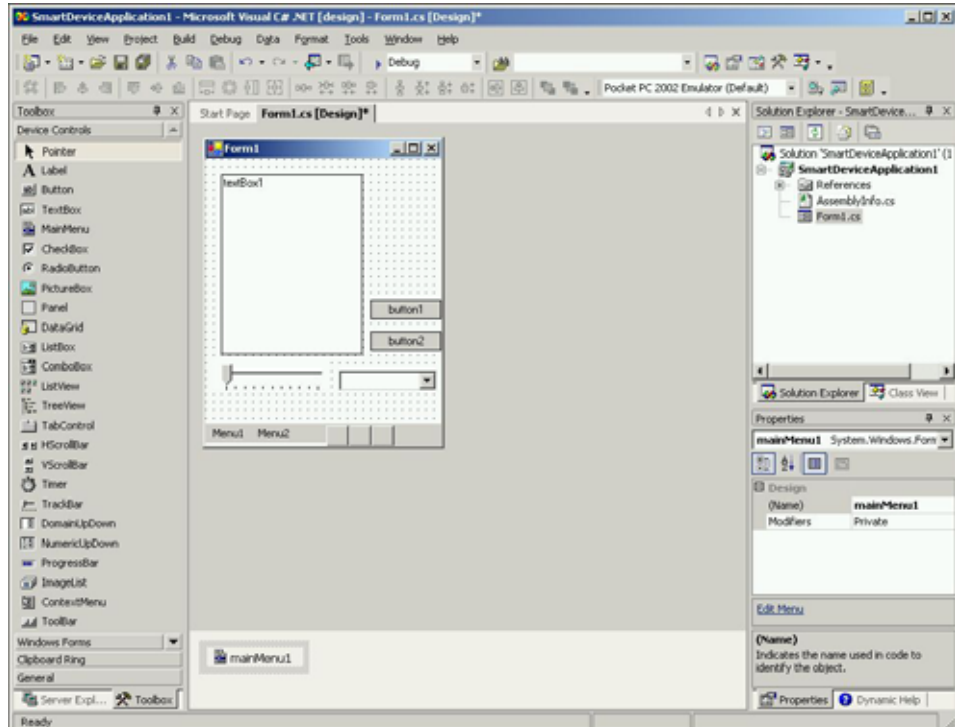


Abbildung 4.4: Der Graphische User Interface Designer der Entwicklungsumgebung Visual Studio mit einer Pocket PC Oberfläche.

(Windows und Mac OS) oder eigene Oberflächenkomponenten implementiert. *Qt* bietet hierzu eigene Konzepte zur Laufzeitunterstützung und Typisierung mit, welche auf die C++ Konzepte aufgesetzt werden. Neben dem Zugriff auf Darstellungsklassen bietet *Qt* auch einen abstrahierten Zugriff auf Dateisystem und Datenbanken. Die Entwicklungsumgebung *Qt Designer* bietet eine interaktive graphische Designumgebung für die Erzeugung von *Qt* Oberflächen. Die Oberflächen können aus dem *Qt Designer* heraus als XML-Beschreibung gespeichert werden. Diese kann dann zur Laufzeit interpretiert werden. Die XML Beschreibung erlaubt die Definition einfacher Layout-Anweisungen und Interaktionen.

Smalltalk *Smalltalk* ist als konsequent objektorientierte Programmiersprache insbesondere bei der Vermittlung und Schulung sehr beliebt. Auch für *Smalltalk* gibt es Laufzeitumgebungen und User Interface Toolkits, welche für die meisten geläufigen Plattformen zur Verfügung stehen. Eine Anzahl weiterer weniger bekannter Lösungen für plattformübergreifende User Interface Toolkits existiert und wird z.B. bei [Pfi02] in einer Übersicht diskutiert.

Zusammenfassung Das Problem toolkitbasierter Ansätze besteht in der Notwendigkeit, einen Toolkit zu definieren, der zum einen abstrakt genug ist um die notwendigen Anpassungen an verschiedene Kontexte zu ermöglichen und zum anderen ausreichend Möglichkeiten bietet, eine qualitativ hochwertige, voll interaktive Benutzerschnittstelle zu spezifizieren. Insbesondere die beiden wichtigsten Ansätze, *Sun Microsystems Java* und *Microsoft .Net* verfolgen hierbei ähnliche Strategien. Sie bieten verschiedenen API für verschiedene Plattformen an, welche bereits zur Designzeit spezifiziert sein müssen. Ob eine plattformspezifische Laufzeit eine Anwendung dann auch interpretieren und darstellen kann wird in beiden Fällen erst zur Laufzeit überprüft.

Während durch klar unterschiedene Entwicklungsplattformen für die verschiedenen Geräteplattformen (J2SE, JPE, J2ME) unterstützt wird, bietet Microsoft einheitliche API (*.Net Framework* und *Compact Framework*) und eine Entwicklungsumgebung an. In beiden Fällen tritt der Konflikt zwischen Optimierung und Kompatibilität klar zu Tage (s. Abschnitt 2.4.3).

Betrachtet man die Charakteristika der hier beschriebenen Toolkits im Hinblick auf die in Abschnitt 2.5 aufgestellten technischen Anforderungen kann folgende Gesamtbewertung gemacht werden:

Plattformunabhängigkeit ist bedingt möglich, da sich in der Zwischenzeit recht einheitliche Toolkit-Bibliotheken herausgebildet haben, die in ihrer Schnittmenge für verschiedene Plattformen (insbesondere in *.Net*) gleichen. Da diese Toolkits meist in Sprachen realisiert sind, die wiederum in einer eigenen Laufzeitumgebung laufen, kann durch Portierung der Laufzeit das UI inklusive der Anwendung auf verschiedene Plattformen übertragen werden. Problematisch ist hierbei allerdings die Abstraktionsebene, die sehr eng an die konkrete Darstellung auf der Plattform orientiert ist und hinsichtlich der Elemente, dem Layout und anderen Gestaltungsaspekten keine Anpassungen zulässt.

Skalierbarkeit ist kein Problem der toolkitbasierten Systeme, da diese relativ nahe an den graphischen Komponenten des Betriebssystems lagern entsteht hier zur Laufzeit wenig Verlust. Toolkitbasierte Systeme existieren heute für fast alle Endgeräteklassen.

Dynamische Erweiterbarkeit Hier liegt einer der großen Vorteile der toolkitbasierten Systeme. Da diese meist eng an die graphischen Systeme angebunden sind, können sie meist Schnittstellen zu diesen bereitstellen, so dass sowohl auf konkreter (Graphik), als auch auf abstrakter (Toolkit) Ebene Erweiterungen durch Zugriff auf die Graphik-API oder durch Erweiterung existierender Elementen möglich ist.

Standardkonformität ist in dem Falle der toolkitbasierten Systeme nicht weit entwickelt. Vielmehr versucht jeder Hersteller solcher Toolkits Entwickler durch spezifische Syntax und Funktionalitäten an sich zu binden. Gleichwohl hat sich aus der Tradition heraus eine gewisse, unverbindliche Konvention des Aufbaus und der Verwendung von Toolkits entwickelt.

In jedem Fall ist das Maß der Anpassung auf der Toolkit-Ebene eingeschränkt. Die atomistische Betrachtungsweise auf der Ebene der Bedienelemente erlaubt es nur in wenigen Fällen Zusammenhänge abzuleiten. So kann die Anpassung auf ein Endgerät schnell zu einer Verstümmelung oder Ver-

fremdung der Darstellung führen weil Elemente die zusammengehören getrennt wurden, der Arbeitsablauf gestört wurde, oder Ähnliches. Eine weitere Ebene der Abstraktion, die auf Anwendungs-, oder auf Dialogebene integriert, scheint daher notwendig.

4.3 User-Interface Management Systeme (UIMS)

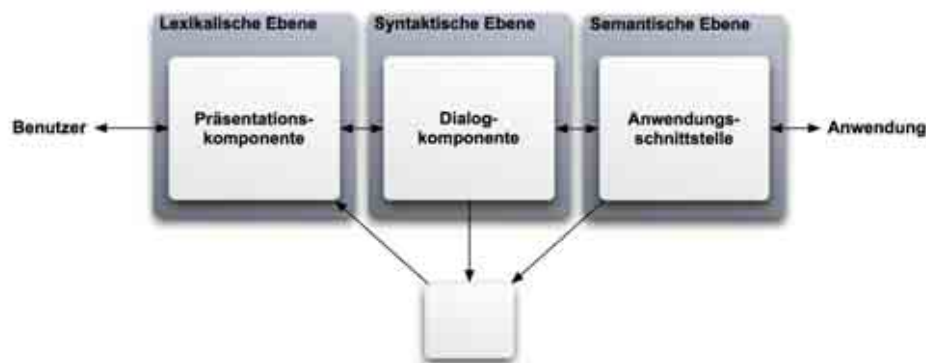


Abbildung 4.5: Die logischen Komponenten eines UIMS im Seeheim Modell nach [DFAB93]

Dix *et al.* [DFAB93] bezeichnen User Interface Management Systeme (UIMS) in erster Linie als konzeptuelle Architektur, bei der die Trennung von Anwendungssemantik und der Anwendungsoberfläche als vornehmliches Ziel gelten. Dieses Konzept der UIMS [Kas82], auch als *Dialogue Management Systems* [RHE⁺82] oder *Abstract Interaction Handlers* [FR82] bezeichnet, wurde entwickelt, um eine höhere Abstraktion der Darstellung zu erhalten [HH89] und so die Entwicklung von Benutzerschnittstellen zu erleichtern und zu beschleunigen [Pfa85, ABDH89, Hüb90]. „Ein User Interface Management System ist ein Werkzeug zur Entwicklung und zur Ablaufsteuerung von Benutzeroberflächen. Das Werkzeug soll die interdisziplinäre Kooperation von rapid prototyping, Management und Fertigstellung von Interaktionen in einem Anwendungsgebiet mit verschiedenen Geräten, Interaktionstechniken und Oberflächenlayouts ermöglichen“ [Sch98]. Da Toolkits meist nur auf den elementaren Ebenen der Interaktoren operieren, wurde nach Methoden gesucht, welche in der Lage sind, eine Anwendung in ihrer Gesamtheit zu betrachten. Abbildung 4.5 zeigt die logischen Komponenten eines UIMS anhand des Seeheim Modells [DFAB93]. Die Kernaufgaben eines UIMS beschreiben Dix *et al.* [DFAB93] wie folgt.

- Darstellung einer konzeptuellen Architektur für die Struktur eines interaktiven Systems, welche auf die Trennung zwischen Anwendungssemantik und Darstellung gerichtet ist.
- Bereitstellung von Techniken, Anwendung und Präsentation getrennt voneinander zu trennen, ohne die Verbindung zwischen den beiden zu verlieren.
- Unterstützung von Techniken zur Verwaltung, Entwicklung und Bewertung eines interaktiven Laufzeitsystems.

Daraus entstehen folgende Vorteile eines UIMS gegenüber der reinen toolkitbasierten Implementierung:

- Anwendungen sind leichter zwischen Systemen zu übertragen (*Portabilität*).
- Sie definieren *wiederverwendbare* Komponenten.
- Eine Anwendung kann einfach mit *verschiedenen* Benutzerschnittstellen kombiniert werden ohne die Anwendungslogik verändern zu müssen.
- Die Anwendung kann einfacher auf die Bedürfnisse der Benutzer angepasst werden (*Personalisierbarkeit*).

Weiterhin können durch die Darstellungsprozesse ergonomische und gestalterische Regeln in Form von Algorithmen umgesetzt werden, und so die Anforderungen an die Entwickler gesenkt werden [Rei00], deren Kenntnis von Guidelines meist eingeschränkt ist [MN90] und die für Unterstützung durch Darstellungssysteme dankbar wären [BHW92].

4.3.1 Spezifikationssprachen

UIMS definieren sich in der Regel dadurch, dass sie spezielle Sprachen zur Beschreibung der User Interfaces verwenden [Mye92]. Myers [Mye92, S. 9 ff.] nennt hierbei folgende Beispiele:

Zustandsdiagramme (State Transition Diagrams) Formale Beschreibung von Zuständen und von Übergängen zwischen diesen, welche durch eine Benutzereingabe ausgelöst werden. Newman [New98] entwickelte sein System, welches als erstes UIMS gilt auf Basis dieses Formalismus. Diese Methode ist geeignet um Systeme zu beschreiben, welche viele verschiedene Modi besitzen. Allerdings ist die Anwendbarkeit auf moderne graphisch-interaktive Systeme nicht sehr gut möglich.

Grammatiken wie die erweiterte BNF (Syngraph-UIMS [OD83]) oder mathematische Formalisierungen (PIE Modell [Dix91]) wurden zur Spezifikation der graphischen Ausgabe genutzt. Aufgrund der Abstraktion dieser Sprachen besteht der Nachteil, dass die Umsetzung meist für die Entwickler problematisch war, da eine Übersetzung in eine graphische Repräsentation meist schwer zu antizipieren war. Neuere Sprachen wie die XML-Schema Definitionen [FW04] stellen ebenfalls eine Grammatik zur Beschreibung von Datenstrukturen dar. Systeme wie Jaxfront [GLS05] wurden entwickelt aus Schema graphische Oberflächen zu generieren.

Ereignissprachen und Produktionssysteme Eine andere Form der Spezifikation von Benutzerschnittstellen setzt auf den Benutzereingaben direkt auf, die in den meisten UIMS in Form von Ereignissen abstrahiert werden. Die Beschreibung erfolgt in diesem Fall durch Regelsysteme. Sie funktionieren nach dem Prinzip: *Wenn eine Mausklick in Element X erfolgt, dann führe Aktion Y aus.* Systeme wie ALGAE [FB87] oder Apple's HyperTalk [App87] bauen auf diesem Prinzip auf. Ein großes Problem dieser Sprachen ist, ähnlich wie bei den Grammatiken, die hohe Abstraktion von der tatsächlichen graphischen Darstellung.

Deklarative Sprachen zeichnen sich dadurch aus, dass sie beschreiben *was* passieren soll und nicht *wie* es passieren soll wie dies für prozedurale Sprachen der Fall ist. Im Gegensatz zu den oben beschriebenen Sprachen wird hier die vor allem die Beziehung zwischen der Anwendung und der Darstellung beschrieben [DFAB93] und weniger die Abfolge von Ereignissen. Deklarative Sprachen sind im Kontext der Entwicklung graphischer Benutzerschnittstellen als Beschreibungen der graphischen Darstellung zu verstehen. Aus diesem Grunde lässt sich diese Form der Spezifikation durch direkte graphisch-interaktive Bearbeitung erzeugen.

Constraint Sprachen beschreiben die Beziehung zwischen Elementen. Das System hat die Aufgabe diese Beziehung über die Laufzeit aufrecht zu erhalten. Nach Dix *et al.* [DFAB93] können *Constraint Sprachen* daher auch als Varianten von deklarativen Sprachen beschrieben werden, die sich jedoch stärker auf die dynamischen Eigenschaften einer Anwendung oder die Beziehung zwischen Werten und Darstellung (*Bindung*) konzentrieren.

Abstrakte Spezifikationssprachen (High-Level) erlauben die Spezifikation auf einer hohen Ebene der Abstraktion, welche weniger mit der endgültigen Darstellung als mit den Konzepten, Aufgaben und Benutzermodellen gemein haben. Ein Beispiel für eine frühe abstrakte Spezifikationssprache ist die IDL von Foley *et al.* [FGK88]. Moderne modellbasierte Konzepte, die häufig auf semantischen Beschreibungen basieren, gehören ebenfalls zu diesen abstrakten Spezifikationssprachen [TCC04, FAPQA04, EVP01, VLM⁺04]. Hier werden verschiedene Ebenen abstrakter Beschreibungen durch iterative Verfeinerung in eine Darstellung überführt.

Objektorientierte Sprachen stellen den Übergang zu den User Interface Toolkits dar, da auch hier auf Basis von Objektmodellen Dialoge spezifiziert werden. Im Gegensatz zu den Toolkits haben die objektorientierten UIMS jedoch die Möglichkeit die Dialoge auf einer höheren Ebene als den Interaktoren zu beschreiben.

Trotz der Vielfalt der Spezifikationssprachen die bislang in UIMS eingesetzt wurden, kann nach Götze [Göt95] ein Vorgehensmodell für die Entwicklung von Benutzerschnittstellen in UIMS beschrieben werden.

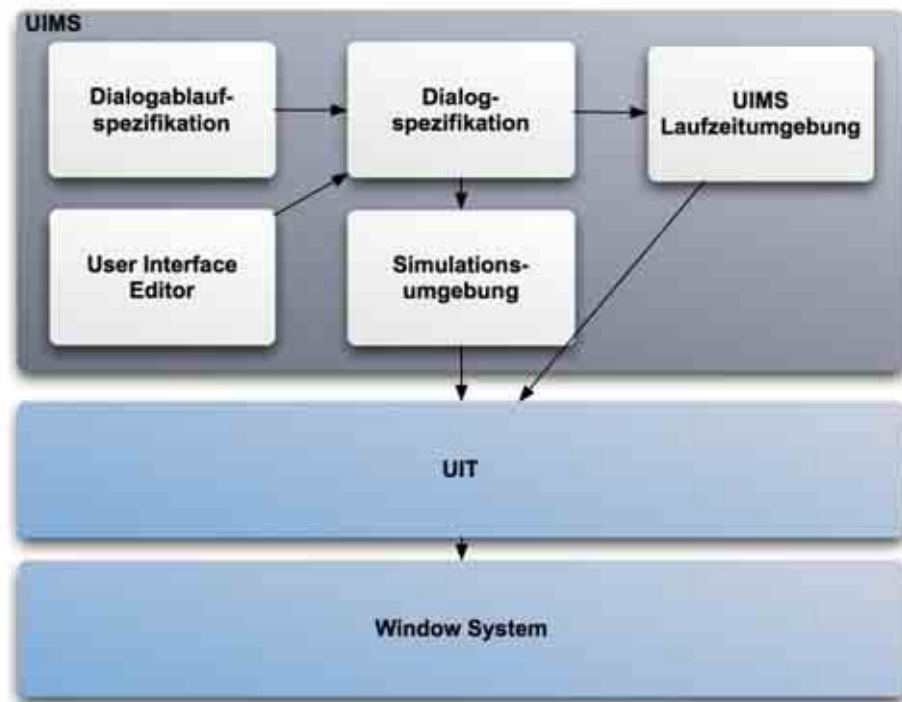


Abbildung 4.6: Entwicklung von Benutzerschnittstellen in einem UIMS nach [Göt95].

Abbildung 4.6 zeigt, wie in einem UIMS über die Spezifikation des Dialogablaufs und die Bearbeitung der Benutzerschnittstelle, auf die Benutzerschnittstelle und deren Verhalten in einem UIMS Einfluss genommen werden kann. Diese Abbildung nach Götze [Göt95] wurde leicht modifiziert, da nach Abbildung 4.1 ein UIMS auf dem User Interface Toolkit aufsitzt. Götze unterscheidet zwischen der Bearbeitung der dynamischen und prozessorientierten Anteile der Benutzerschnittstelle, die über Ereignissprachen, Zustandsdiagramme oder Ereignissprachen beschrieben werden und über ein Werkzeug zur Beschreibung des Dialogablaufs bearbeitet werden. Dagegen stehen die darstellungsorientierten und statischen Aspekte, die durch deklarative Sprachen beschrieben werden und über einen (graphischen) User Interface Editor bearbeitet werden.

4.3.2 Spezifikation durch Modelle

Der zentrale Aspekt von UIMS ist die Trennung der verschiedenen Ebenen der Mensch-Maschine-Kommunikation in logisch unabhängige Bereiche. Verschiedene Aspekte können so unabhängig spezifiziert und gegebenenfalls ausgetauscht werden um die Darstellung an neue Anforderungen des Systems, des Kontextes oder des Benutzers anzupassen. Oben wurde auf die verschiedenen Sprachen eingegangen, die zur Spezifikation von Benutzerschnittstellen in UIMS genutzt werden können. Jeder Formalismus, so wurde gezeigt, hat seine Vorteile bezüglich bestimmter Aspekte eines interaktiven Systems.

Unterschiedliche UIMS haben aus diesem Grunde auch verschiedene Aspekte der Darstellung und verschiedene Abstraktionsebenen, verbunden mit den dafür geeigneten Formalismen, eingesetzt. Daraus haben sich komplexe mehrschichtige UIMS entwickelt, welche inkrementell die Generierung von Oberflächen auf Basis einer schrittweisen Konkretisierung abstrakter Spezifikationen betreiben [Sze96]. Für diesen Typ der komplexen mehrschichtigen UIMS hat sich der Begriff der *modellbasierten UIMS (MBUIMS)* [Mye95] und *modellbasierte User Interface Development Environments*

(*UIDE*) [Pin00] etabliert. Tatsächlich erscheint die Trennung in *UIMS* und *MBUIMS* willkürlich, betrachtet man die Tatsache, dass die Berücksichtigung verschiedener Modellierungsebenen Teil dieses Konzept darstellen und die Abstraktion stets ein Modell des zu Abstrahierenden Aspektes voraussetzt.

„[Ein Modell ist eine i]dealisierte, vereinfachte, in gewisser Hinsicht ähnliche Darstellung eines Gegenstandes, Systems oder sonstigen Weltausschnitts mit dem Ziel daran bestimmte Eigenschaften des Vorbilds besser studieren zu können.“ [Sch98]

Ist ein UIMS modellbasiert wenn es lediglich eine Abstraktion der Darstellung nutzt, oder ist für die Bezeichnung *modellbasiert* ein höherer Abstraktionsgrad notwendig? Ein Unterscheidungskriterium erscheint hier willkürlich und wenig zielführend, zumal einige Ansätze, wie UIML [FAPQA04, AP99] oder auch der *Universal Remote Console Standard* [Int05a] eine Entwicklung hin zu abstrakteren Modellen wie z. B. Aufgabenbeschreibungen erfahren. Da Silva [Pin00] schlägt daher ein Kontinuum zur Einordnung von UIMS anhand ihrer Komplexität vor.

Das Prinzip modellbasierter UIMS und UIDE ist es, die Oberflächenspezifikation auf der inhaltlichen Ebene der Anwendung (Aufgaben, Benutzerrollen, Interaktoren, etc.) zu beschreiben. Ziel der Oberfläche ist es ja lediglich diese Anwendung oder Aufgabe dem Nutzer zugänglich zu machen. Durch diese Abstraktion soll die Fokussierung auf die graphische Darstellung bei der Oberflächenentwicklung überwunden und die Einbringung anderer Aspekte unterstützt werden. Die Generierung verschiedener Darstellungen kann theoretisch wesentlich besser und flexibler sein wenn dabei die eigentliche Aufgabe und die zugrundeliegenden Konzepte nutzbar sind.

Puerta et al. [PCOM99] argumentieren, dass modellbasierte Ansätze den Fokus wieder auf die eigentliche Aufgabe, welche mit der Anwendung ausgeführt werden soll, richten. Domänenwissen von Experten kann in Form von modellierten Konzepten in die Oberflächengestaltung einfließen. Der Benutzer und dessen Aufgabe können ebenfalls modelliert werden und rücken damit in Vordergrund (user-centered design [ND86]). Eine ausführliche Übersicht modellbasierter Ansätze findet sich in den Arbeiten von da Silva [Pin00] und Myers [Mye95, MHP00], sowie in [Sze96, Van97, Sch96].

Da Silva [Pin00] gliedert die existierenden MB-UIMS historisch in drei Generationen.

Die erste Generation fokussierte sich auf die Beschreibung von abstrakten Spezifikationssprachen, wie Petri-Netzen, Regeln, Constraints, etc. und weniger auf die Umsetzung in lauffähigen UIM Systemen. Diese waren meist auf eine Form der Spezifikation ausgerichtet und wenig flexibel. Die Unterstützung für den Entwicklungsprozess war gering.

Die zweite Generation der UIMS baute auf die erste Generation auf indem sie vor allem die Ausführung der Spezifikationen und deren Entwicklungsprozess weiterentwickelten. Hier war wiederum der Grad der Abstraktion gering, d. h. die Beschreibung konkreter Gestaltungsmerkmale wie der Interaktoren traten bereits früh im Gestaltungsprozess auf. Beispiele für solche Systeme der zweiten Generation sind nach Da Silva [Pin00]:

- *COUSIN* [HSL85]
- *UIDE* [FCKKM91, KF90]
- *HUMANOID* [LSN93, SLN92]
- *ITS* [WBBG90, WB90]

Die dritte Generation von UIMS unterscheiden sich von ihren Vorgängern in erster Linie dadurch, dass die Spezifikation in Form abstrakter Modelle und in mehreren Ebenen erfolgt. Beispiele für UIMS dieser Generation sind:

- *ADEPT* [MPWJ92]
- *MASTERMIND* [BDRS97, SSC⁺95]
- *FUSE* [LS96]
- *MECANO* [Pue96], *MOBI-D* [PE98] und heute *XIML* [PE04, PE02]
- *TADEUS* [ES95]
- *Teallach* [GBM⁺99]
- *TRIDENT* [BHL94] und heute *UsiXML* [LV04, LVM⁺04, VLM⁺04]
- *TERESA* [BCMP04] und *CTTE* [MPS02, PMM97, Pat99]

Tatsächlich stehen am Anfang der Entwicklung modellbasierter UIMS, jene benutzerzentrierten Ansätze, welche auf Basis von Aufgabenmodellen besonders an die Aufgabe angepasste Oberflächen zu generieren suchten: „The task model is the knowledge source where the UI is described in the user's own words, not in terms of the system or the designer.”[WJ95]. Vanderdonck *et al.*[VP99] argumentieren, dass die besondere Bedeutung von Aufgabenmodellen zum einen darin liegt, dass diese meist als Basis zur Entwicklung weiterer Modelle dienen, und zum anderen darin, dass ein Aufgabenmodell die Interessen des Nutzers, also die einfache Erfüllung der Aufgabe, in den Mittelpunkt des Designprozesses stellt.

Weiterhin steht als wesentliche Motivation der modellbasierten UIMS natürlich die Möglichkeit der automatischen Generierung von Benutzungsoberflächen für verschiedene Plattformen, Modalitäten oder Nutzer oder generell die kontextabhängige Erzeugung von Oberflächen auf Basis einer gemeinsamen Spezifikation im Vordergrund. Alle Darstellungen werden aus derselben Ressource generiert. Dadurch werden sowohl Entwicklungs- als auch Wartungsaufwand minimiert. Vanderdonck [VP99] stellt eine Übersicht der Modelle dar, welche bei der Entwicklung von Benutzerschnittstellen verwendet werden.

Mit der Anzahl der verwendeten Modelle steigt natürlich auch die Komplexität und der Aufwand bei der Erzeugung und Umsetzung der endgültigen Darstellung. Schlanke Lösungen verzichten auf komplexe Benutzer- oder Kontextmodelle (UIML), umfassende Ansätze versuchen alle Daten bei der Erzeugung von Benutzerschnittstellen zu berücksichtigen (XIML, ARTStudio) [PE04, TCC04].

Modellbasierte Ansätze benötigen mehrere Prozeßschritte um aus den zugrundeliegenden Modellen zunehmend konkretere Beschreibungen und letztlich die Darstellung der Oberfläche zu erreichen. Ziel dieser Ansätze ist es, eine weitgehende Automatisierung des Darstellungsprozesses zu erreichen. Transformationsregeln zwischen den verschiedenen Abstraktionsebenen und intermediären Modellen müssen definiert werden. In vielen Fällen ist diese Transformation jedoch nur halbautomatisch unterstützt oder gänzlich manuell. Die Transformation kann also zur Laufzeit, zur Designzeit oder auch nur teilweise zur Designzeit erfolgen.

Die Vorgehensweise ist hierbei, bestimmte Zwischenschritte als Teil des Designprozesses durchzuführen und die so angepassten Zwischenversionen zu speichern. Basierend auf diesen Zwischenversionen werden dann zur Laufzeit die endgültigen Darstellungen generiert (z.B. UIML, XIML). Thevenin *et al.*[TCC04] schlagen die horizontale Übersetzung (*translation*) zwischen verschiedenen Anwendungsfeldern als eine zusätzliche Dimension vor. *Reverse-Engineering* Methoden erlauben weiterhin die Extraktion von Modellen aus existierenden Oberflächen und öffnen somit die umgekehrte Entwicklungsrichtung [BVS02, GBL03].

Angesichts der technische Anforderungen, die in Abschnitt 2.5 beschrieben wurden, erscheinen UIMS geeignetsten, diese zu erfüllen, wie die folgende Aufstellung zeigt.

Plattformunabhängigkeit ist aufgrund der Abstraktion möglich. Eine Abtrennung der Darstellung von der Implementierung über eine abstrakte Spezifikation öffnet die Möglichkeit plattformspezifische UIMS-Laufzeiten zu realisieren, die die Spezifikation umsetzen. Je umfassender die Spezifikation ist, und je weiter syntaktische und semantische Aspekte der Benutzerschnittstelle mit berücksichtigt werden, desto stärker kann hierbei von der Plattform abstrahiert werden.

Skalierbarkeit ist gegeben, da auch hier eine Anpassung in der Regel aufgrund der Laufzeitumgebung möglich ist. So lassen sich verschiedenen Anwendungsszenarien denken in denen der Einsatz möglich ist.

Dynamische Erweiterbarkeit muss in einem solchen System meist an allen Stellen des Systems erfolgen, je stärker die Abstraktion, desto mehr Prozessebenen müssen daher angepasst werden. Aus diesem Grund muss hier zwischen Abstraktion und Erweiterbarkeit abgewägt werden.

Standardkonformität ist in manchen Fällen bereits Realität, so sind einige der User Interface Beschreibungssprachen wie die *Hypertext Markup Language (HTML)* oder *XForms* [DKMR03] bereits durch internationale Gremien wie das W3C standardisiert worden. Auch die *UIML* Sprache [Pha00] wurde bei Oasis Open [Oas05] zur Standardisierung eingereicht. Der *Universal Remote Console* [Int05b] Standard wurde bei der ANSI standardisiert.

Modell	Inhalt	Darstellung
Aufgabenmodell(task)	Was der Benutzer macht und weshalb	Aufgaben werden entweder hierarchisch als Sequenzdiagramme dargestellt, wobei der Schwerpunkt auf der Abfolge und den notwendigen Vorbedingungen liegt, oder als Prozess- bzw. Datenorientiertes Modell. (z.B. ConcurTaskTree, [PBSK99])
Domänenmodell(domain)	Konzepte, Begriffe, Handlungen	Domänenmodelle beschreiben die Konzepte und Begriffe, welche in einem Anwendungsfeld bekannt sind. Dieses Modell entspricht weitestgehend dem Begriff der Ontologie. [NFF ⁺ 91]
Dialog(dialogue)	Struktur des Dialoges zw. Mensch und Maschine	Abfolge von Interaktionskomponenten und Dialogen in Abhängigkeit von der Benutzereingabe.
Konkrete Interaktion — Präsentation(presentation)	Darstellung und Struktur der Ausgabe	Repräsentation der abstrakten Interaktoren, bzw. Komponenten des UI
Konkrete Interaktion — Verhalten(behavior)	Dateneingabemöglichkeiten und Interaktion	Beschreibt die Interaktionsmöglichkeiten über die der Benutzer den Programmfluss steuern kann. Kontrolle(control) Dienste und Funktionen der Anwendung Liste der verfügbaren Funktionen oder Dienste welche über das UI angesprochen werden können.
Plattform(platform)	Gerätebeschreibung	Beschreibung der Ein- und Ausgabemöglichkeiten und sonstiger Eigenschaften des Endgerätes
Umgebung (environment)	Kontextuelle Informationen	Physikalische oder soziale Umgebungsparameter wie Ort, Lautstärke, Kultur, etc.
Benutzer(user)	Eigenschaften des Benutzers	Mehr oder weniger strukturierte Klassifikation von Benutzer Eigenschaften (user modelling)

Tabelle 4.1: Übersicht der verwendeten Modelle nach Vanderdoct, zitiert nach [Tra02].

In diesem Abschnitt werden existierende UIMS betrachtet und anschließend hinsichtlich der Anforderungen bewertet werden. Im Folgenden soll vor allem auf die *Beschreibungssprachen* (Abschnitt 4.3.3.1) eingegangen werden, da diese meist an der Grenze zwischen deklarativen und abstrakten Sprachen stehen, sowie auf die *modellbasierten Ansätze*, die Benutzerschnittstellen meist auf mehreren Ebenen der Abstraktion spezifizieren.

4.3.3 Existierende modellbasierte Ansätze

Die existierenden Ansätze, welche in diesem Abschnitt beschreiben werden, unterscheiden sich in erster Linie durch die Art der Modelle, welche bei der Spezifikation der Benutzerschnittstelle verwendet werden. Die in diesem Abschnitt beschriebenen Ansätze steigen in ihrer Komplexität an, so sind die in Abschnitt 4.3.3.1 beschriebenen Beschreibungssprachen in der Regel, auf die konkreten und abstrakten Darstellungsmodelle beschränkt. Das *ITS* [WBBG90] (siehe Abschnitt 4.3.3.2) bietet zusätzlich eine geringe Unterstützung bei der Spezifikation von Aufgaben.

Trident und *UsiXML* sind eine Familie von Werkzeugen, die aufbauend auf dem *CAMELEON* Modell in den Arbeitsgruppen um Vanderdoct [LVM⁺04] und Coutaz [TCC04] entwickelt wurden. Diese werden in Abschnitt 4.3.3.3 beschrieben. Weiterhin wird ein Ansatz aus der Arbeitsgruppe um Paternó [MPS04] beschrieben. Zuletzt wird die *eXtensible Interface Markup Language (XIML)* von Puerta und Eisenstein [PE02] behandelt. Diese steht in einer langen Tradition modellbasierter Ansätze wie *MECANO* [Pue96] und *MOBI-D* [PE98].

Zu erwähnen sind an dieser Stelle noch andere Konzepte modellbasierter UIMS, die meist in speziellen Anwendungsszenarien eingesetzt werden wie agentenbasierte Benutzerschnittstellen als Kommunikationsschnittstelle zwischen Software-Agenten und dem Benutzer, migrierbare User Interfaces [BS03a, BS03b, BB03] die zwischen verschiedenen Endgeräten wechseln und dabei den Zustand behalten können, oder speziell webbasierte Ansätze wie *AutoWeb* [FP00], *XWeb* [OJNF00], *WebML* [BCFM01, CFB00] oder *Personal Interfaces2Go* [TDE03].

Die Auswahl der Ansätze richtet sich in erster Linie nach ihrer Aktualität, und der Bedeutung, die diese Ansätze für die Problemstellung der plattformübergreifenden Entwicklung haben. Einige Ansätze, die hier nicht ausführlicher erwähnt werden können, sind *Dygimes* [CLC04a, CLV⁺03, BLC03], *Multi-Device Authoring Technology* [BBC⁺04], *CHIRON* [TNB⁺95] und *PALIO* [SPZ04]. Eine Übersicht weiterer Ansätze findet sich in [RBG00, SJ04b].

4.3.3.1 Beschreibungssprachen

Die Spezifikation mit Hilfe von textbasierten Beschreibungssprachen hat zu Beginn des neuen Jahrtausends, mit dem Durchbruch der *eXtensible Markup Language (XML)* und den damit verbundenen Technologien eine wahre Renaissance erlebt. Allein auf den *Cover Pages* findet sich eine Übersicht von ca. 20 Anwendungen von XML zur Beschreibung von User Interfaces [Cov05]. Neue Technologien für Benutzerschnittstellen werden in naher Zukunft verstärkt auf diese Form der Darstellungsspezifikation aufsetzen.

Microsoft hat im Jahr 2007 eine grundlegend überarbeitete Benutzerschnittstelle namens *Aqua* für sein Betriebssystem Windows veröffentlicht. In diesem von Grund auf überarbeiteten System mit dem Namen *Vista* können Oberflächen mittels der *Microsoft Extensible Application Markup Language (XAML)* beschrieben und in eine externe XML-basierte Ressource gespeichert werden [Koc05]. Anwendung und Darstellung können so stärker entkoppelt werden. Diese Konzepte wurden bereits in anderen Ansätzen, wie z.B. der *User Interface Markup Language (UIML)* [AP99, Pha00, FAPQA04] entwickelt und sind nun in der kommerziellen Anwendung übernommen. Eine ausführliche Zusammenfassung aktueller XML-basierter Beschreibungssprachen findet sich auch bei Souchon und Vanderdoct [SV03]

User Interface Markup Language (UIML) Die *User Interface Markup Language (UIML)* [AP99, Pha00, FAA01] ist eine deklarative Beschreibungssprache auf Basis von XML. Sie wurde in

Version 3.0 durch ein gemeinsames Aufgabenmodell erweitert. In UIML wird eine Oberfläche über die Struktur der enthaltenen Elemente (*structure*), die Darstellungseigenschaften (*style*), die Interaktion (*behavior*) und eine Abbildung auf ein konkretes Vokabular (*peers*), in welchem die abstrakten Elemente auf konkrete Klassen eines Toolkits abgebildet werden, beschrieben.

Aufgrund der engen Bindung der Sprache an konkrete Darstellungen, v. a. in den *style*- und *peers*-Eigenschaften, kann mittels UIML eine Anwendungsoberfläche sehr gut und detailliert beschrieben werden. Allerdings führt dies zu dem Nachteil, dass die Plattformunabhängigkeit nur sehr eingeschränkt innerhalb einer UIML-Ressource gegeben sein kann [Ric02b]. Die Struktur, Darstellung, Interaktion und das Vokabular lassen sich nicht ohne weiteres auf verschiedene Plattformen übertragen [FAPQA04]. Dennoch kann dieselbe UIML-Ressource für verschiedene Darstellungen auf derselben Geräte-Familien herangezogen werden (Desktop Familie, PDA Familie und WAP Familie).

Aus diesem Grunde wurde in der Version 3.0 der UIML Sprache eine weitere Abstraktionsebene eingeführt: das Aufgabenmodell (*task model*). Die Notation basiert auf der *Concurrent Task Tree* Notation von Paterno [Pat99]. Aus dem Aufgabenmodell werden dann verschiedene UIML-Ressourcen für verschiedene Gerätefamilien sowie die Navigationsstruktur erzeugt. Aus dem UIML werden dann wiederum die Darstellung für die verschiedenen Plattformen erzeugt.

Die Darstellung selbst wird durch Darstellungsmodule, sog. *Renderer* realisiert, welche das UIML in eine Darstellung umsetzen, entweder indem sie UIML in HTML umwandeln, welches dann in einem Browser angezeigt werden kann, oder indem ein *Renderer* direkt eine Darstellung erzeugt, in Form z.B. eines Java Swing Dialoges. UIML selbst erlaubt die Einbindung plattformspezifischer Eigenschaften, so dass eingeschränkte Optimierungsmöglichkeiten gegeben sind.

Die Anpassung an die Plattformen wird über das Vokabular realisiert. Je generischer ein Vokabular aufgebaut sind, desto breiter ist das Spektrum der adressierbaren Plattformen, derzeit sind ein generisches Vokabular für Java und HTML bzw. für WML-Dialekte verfügbar [FAPQA04].

Die Entwicklung von Anwendungsoberflächen in UIML erfolgt in einem transformationsbasierten Prozess, d. h. das abstrakte Modell wird in mehreren Schritten in die konkrete Darstellung transformiert. Dieses Vorgehen birgt das Problem in sich, dass der Entwickler über mehrere Schritte hinweg die Folgen seiner Designentscheidungen antizipieren muss. Ali [FAPQA04] spricht hier von einem Prozess des *Trial and Error*. Diese Art des Vorgehens ist sehr aufwändig und fehleranfällig.

Mit Hilfe der *Transformation-based Integrated Development Environment (TIDE)* kann der Entwickler die Ergebnisse der Änderungen im UIML-Code direkt in einer graphischen Vorschau überprüfen. TIDE bietet einen hierarchischen Editor in dem der UIML-Code bearbeitet werden kann. Über einen Knopfdruck wird dieser Code dann von einem Darstellungsmodul zur Anzeige gebracht. Die Korrespondenzen der graphischen Elemente mit den Bereichen im Code werden über Linien hervorgehoben. Während TIDE noch kein generisches Vokabular und auch keine Task-Modelle unterstützt, werden in der angekündigten Version TIDE2 diese Aspekte adressiert werden.

Mit UIML hat die Forschergruppe von Abrams einen leistungsfähigen Ansatz entwickelt, der weitreichende Anerkennung gefunden hat. Die konzeptionelle Modellierung der Oberfläche in Struktur, Darstellung, Verhalten und Abbildung ist einleuchtend und erlaubt ein hohes Maß an plattformspezifischer Optimierung, so dass auch komplexere Layouts auf verschiedene Plattformen gebracht werden können. Die Möglichkeit konkrete Methoden an Elemente zu binden (Verhalten) erlaubt die Beschreibung einer kompletten interaktiven Benutzungsoberfläche und der Stand der Implementierung war bereits sehr früh (2000) sehr weit fortgeschritten, so dass auch ausreichend praktische Erfahrungen mit diesem Ansatz gesammelt werden konnten. Allerdings hat die Entwicklung von UIML bis zur heutigen Version 3.0 auch gezeigt, dass diese Modellierung nicht ausreichend war, um wirkliche Plattformunabhängigkeit zu gewährleisten. Die Einführung einer zusätzlichen Abstraktionsebene scheint zwar klar motiviert, aber auch konzeptionell nicht schlüssig, was letztlich zu zusätzlichem Entwicklungsaufwand führt.

XForms - Dubinko, W3C *XForms* ist ein Standard für elektronische Formulare. Dieser wurde vom W3C in der Version 1.0 freigegeben wurde [Dub03, DKMR03], aktuell existiert ein Entwurf in der Version 1.1 [Boy05]. Ziel dieser neuen Formulare ist es, die traditionellen HTML Formulare,

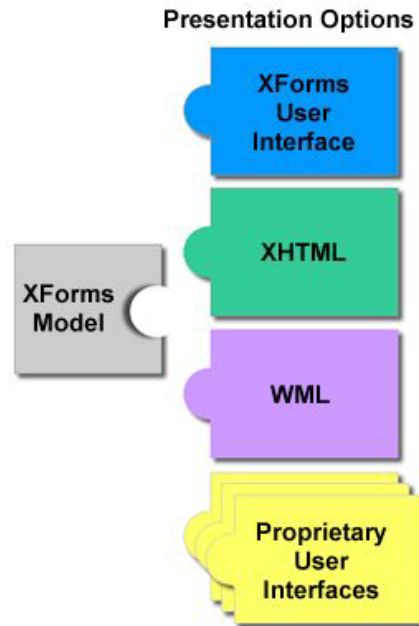


Abbildung 4.7: Die Architektur von XForms [DKMR03] ermöglicht es, dass verschiedenen Benutzungsschnittstellen mit dem XForms Model arbeiten können.

abzulösen. Da bei modernen Webanwendungen immer umfangreichere Transaktionen benötigt werden, von denen viele den Rahmen der ursprünglichen Formulartechnik sprengen und Funktionen wie Validierung und Serialisierung häufig neu entwickelt werden müssen. Hierbei handelt es sich nicht um ein komplett eigenständiges Format, sondern es ist zur Integration in andere XML-Sprachen wie XHTML oder SVG vorgesehen.

Bei XForms werden die Regeln zur Beschreibung, Validierung und Präsentation von Formularen in XML ausgedrückt. Dies gilt auch für die bereitgestellten Daten. So schafft XForms die Basis für Verbindungen mit anderen XML-Anwendungen. Es trennt bewusst den Inhalt eines Formulars von seiner Repräsentation und erreicht mit dieser Trennung einen hohen Grad der Geräteunabhängigkeit, indem sie die Autoren von Webanwendungen in die Lage versetzt, das Datenmodell einmal für alle Geräte zu schreiben. Da das Datenmodell nicht an die Präsentation gebunden ist, können die Entwickler die Präsentation optimal an die Benutzungsschnittstelle eines jeden Geräts anpassen.

Durch die Integration von Ereignissen, Validierung von Benutzereingaben und einen größeren Funktionsumfang in den Standard, kann in einer Webanwendung der Client autonom viele Aufgaben erledigen. Das verringert die Zahl der Serverzugriffe und verbessert das Antwortverhalten von Anwendungen. Zusätzlich reduziert es die Notwendigkeit von Skripten, indem der Ersteller der Formulare beispielsweise direkt Abhängigkeiten zwischen Feldern angibt, wodurch sich Scripte erübrigen.

Neben den üblichen Paaren aus Namen (*key*) und Werten (*value*) kann eine XForms-Implementation das Resultat eines Formulars außerdem in Form eines XML-Dokumentes übermitteln. Dieses kann somit sehr einfach maschinell weiterverarbeitet werden, was den Aufwand an die Entwicklung von Anwendungen verringert und eine einfache Anbindung an bestehende Systeme ermöglicht.

Die Architektur von XForms sieht es vor, dass eine alleinstehende geräteunabhängige Datenspezifikation, das *XForms Model*, die Fähigkeit hat, mit verschiedenen Oberflächenspezifikationen in HTML oder anderen Sprachen zusammenzuarbeiten. Das *XForms User Interface* ist die Benutzungsschnittstelle von XForms und beinhaltet eine Sammlung von Bedienelementen die dazu dienen, die Formularelemente von HTML zu ersetzen. Mit diesen Bedienelementen kann jedoch keine eigenständige Schnittstelle erzeugt werden. Sie werden direkt in anderen XML-Sprachen wie XHTML oder SVG eingebettet. Unabhängig von den vordefinierten Bedienelementen können auch eigenständige Bedienelemente entwickelt werden, die z.B. für die sprachgesteuerte Eingabe mit dem *XForms Model* zusammenarbeiten.

Universal Remote Console (V2) Die *Universal Remote Console (URC)*, entwickelt vom V2 technischen Komitee des *InterNational Committee for Information Technology Standards (INCITS)*, bezeichnet eine Möglichkeit zur Spezifikation verschiedener Aspekte adaptierbarer, mobiler Benutzerschnittstellen und deren Kommunikation mit entfernten Anwendungen. Hauptanwendungsfeld dieses Ansatzes, der in als *American National Standard - Information Technology ANSI/INCITS 389ff.* verabschiedet wurde, ist die Anbindung mobiler Bediengeräte (Remote Consoles) an informationstechnische Installationen (*target devices*), Endgeräte aus der Unterhaltungselektronik, Terminals, etc. [ZVG02, NM04, Int05b, Int05e, Int05d, Int05a, Int05c].

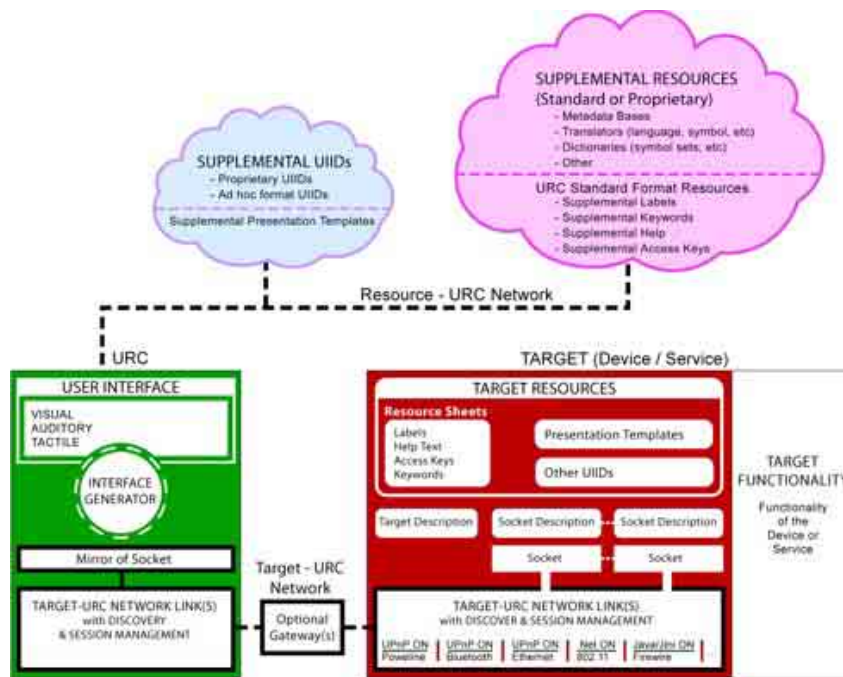


Abbildung 4.8: Architektur des V2 Standards nach [Int05a].

Abbildung 4.8 ist ein Ausschnitt aus dem Standard (ANSI/INCITS 389ff.) [Int05a], welcher die Architektur einer Standard-Plattform beschreibt. Die Kommunikation zwischen einem clientseitigen Mobilgerät (*Universal Remote Console, URC*) und den anzusprechenden Zielgeräten (*Targets*) erfolgt über den Austausch von XML-basierten Beschreibungen. Diese werden als *User Interface Sockets* abstrahiert. Die Beschreibung kann modular aus verschiedenen Teilen aufgebaut werden und so eine Beschreibung der Benutzerschnittstelle, der aufzurufenden Methoden, Ressourcen und Inhalte kommuniziert werden.

Der ANSI-Standard „*Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents*“ umfasst folgende Teile:

- INCITS 389 [Int05a]: Universal Remote Console. Dieser Teil beschreibt die grundlegende Architektur des Standards.
- INCITS 390 [Int05b]: User Interface Socket Description. Dieser Teil beschreibt die Protokolle zwischen Targets (Endgeräten) und benutzerseitigen Geräten.
- INCITS 391 [Int05c]: Presentation Templates. Beschreibung der Darstellungseigenschaften, inkl. textueller Inhalte.
- INCITS 392 [Int05d]: Target Properties Sheet. Beschreibung der Eigenschaften der Endgeräte.
- INCITS 393 [Int05e]: Resource Description. Beschreibung der notwendigen Ressourcen.

Gegenwärtig ist die Erweiterung des V2 Standards um eine Aufgabenbeschreibung, durch die die Fähigkeiten des Targets beschrieben werden können, im Gespräch [Zim05]. Abbildung 4.9 zeigt eine Demonstration eines V2-Systems.

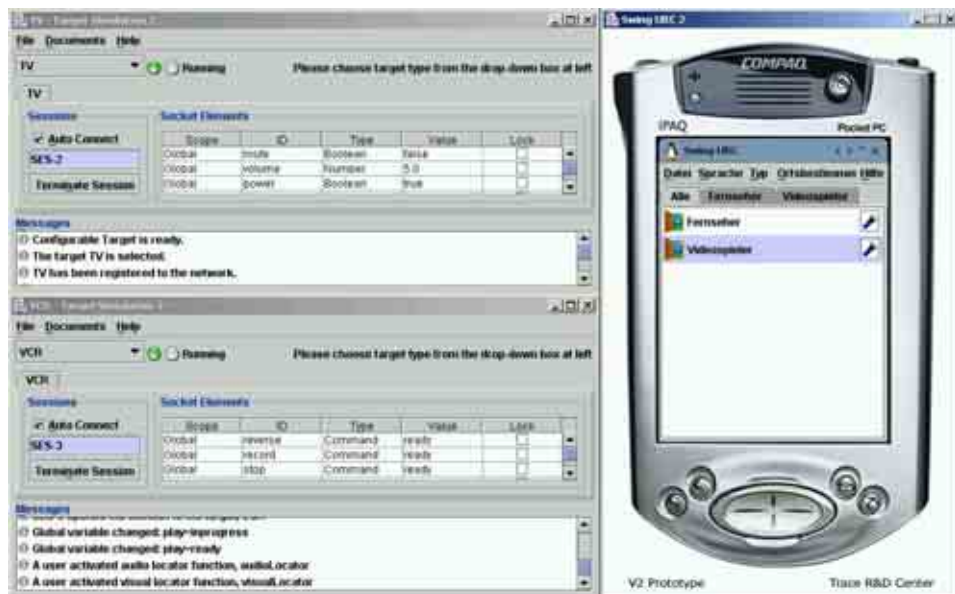


Abbildung 4.9: Beispielimplementierung des ANSI/INCITS 389 ff. des TRACE-Center.

Im Jahr 2005 begann das von der Europäischen Union geförderte Projekt *I2HOME* (Projekt Homepage: <http://www.i2home.org>, welches die Umsetzung des V2 Standards zur Unterstützung kognitiv behinderter und älterer Menschen im Umgang mit Haushaltselektronik zum Ziel haben wird [ARZ⁺05, ARB06].

4.3.3.2 ITS

Die zur Spezifikation der Darstellung und der Stilregeln in dem ITS-System [WBBG90] entwickelte Beschreibungssprache kann als eine der frühen deklarativen Beschreibungssprachen gewertet werden. Das ITS-System, welches auf der EXPO'92 in Sevilla, Spanien eingesetzt wurde, ist ein klassisches UIMS, welches versucht, die Darstellung und die Anwendungslogik zu trennen.

Die ITS-Architektur besteht aus vier Ebenen: einer *Aktions*-Ebene, welche den Datenaustausch zwischen der Anwendung und der Oberfläche mittels Datentabellen und Ereignissen regelt und die Daten über verschiedenen Darstellungen hinweg synchronisiert, einer *Dialog*-Ebene, welche die Aufteilung von Eingaben in *Frames*, sowie den Kontrollfluss spezifiziert, einer *Regel*-Ebene in welcher die Stilregeln zur Darstellung vorgehalten und angewandt werden, und letztlich der darstellenden *Stil*-Ebene auf der die Darstellungsbeschreibung interpretiert wird und ein endgültiges Layout gemacht wird.

Die Repräsentation des Dialoges basiert im wesentlichen auf einem gruppierenden *Frame*-Element, einem *Form*-Element als Rahmen für Einzelfelder, einem *List*-Element als Rahmen für wiederholte Strukturen, und einer Auswahl (*Choice*), deren Inhalte jeweils an Datenfelder der Action-Ebene gebunden sind.

Die Darstellung basiert auf der Anwendung von Stilregeln, welche zur konsistenten Umsetzung der Dialogstruktur in eine Darstellung dienen sollen [WB90, BBGG89]. Diese Stilregeln werden angewendet, wenn die in der Regel angegebene Vorbedingung erfüllt ist. Als Ergebnis der Anwendung der Stilregeln kann die Dialogstruktur umgestellt werden, Darstellungsattribute angepasst werden oder Inhalte erzeugt werden. Auch die Verkettung von Regeln ist möglich.

Die Stilregeln erlauben eine detailliertere Kontrolle der Darstellung als andere UIMS, und sie ermöglichen eine konsistente oder als konsistent wahrgenommene Abbildung des Dialogs über verschiedene Anwendungen hinweg. Die so erzeugte Darstellungsbeschreibung kann dann von der Stilebene umgesetzt werden, wobei das automatisierte Layout der Komponenten auf der verfügbaren Fläche mit dem Ziel der optimalen Ausnutzung der Oberfläche erzeugt wird.

Die Frage der generellen Konsistenz solcher regelgenerierten Darstellungen wurde von Grudin [Gru92] angezweifelt, da diese gegen manche semantische Einschränkungen nicht sensibel sind (s. auch [Gru89])[WBBG90, S.218 u. 222].

Listing 4.1: Beispiel für eine Style-Regel in ITS

```

1 //Regel gilt für alle choices}
2 :conditions source=choice
3 // Beispiel für ein Standard-Attribut für das Titel-Element}
4 :Attributes type=Title, Font=TimesRoman, LeftMargin=10
5 //Es wird eine Vertikale Tabelle eingerichtet}
6 :unit type=VertGroup
7   // Ein Titel wird eingesetzt}
8   :unit type=Title
9   :eunit
10  // Eine Horizontale Tabelle wird eingerichtet}
11 :unit type=HorzGroup
12 //Für jedes Element in der Choice wird ein Element eingefügt}
13 :unit type=Message, replicate=all
14 :eunit
15 :eunit
16 :eunit
17 :econditions

```

4.3.3.3 USIXML, TERESA und das CAMELEON Referenzmodell

Das CAMELEON Projekt (EU IST, 2001-2004) [CCB⁺03] vereint die führenden Arbeitsgruppen Europas auf dem Gebiet der modellbasierten Entwicklung und multimodaler Benutzerschnittstellen in dem Exzellenznetzwerk SIMILAR (<http://www.similar.cc>). Ziel des Projektes ist es einen gemeinsamen Ansatz der modellbasierten User Interface Entwicklung zu formulieren und den Austausch an Werkzeugen und Methoden zu fördern.

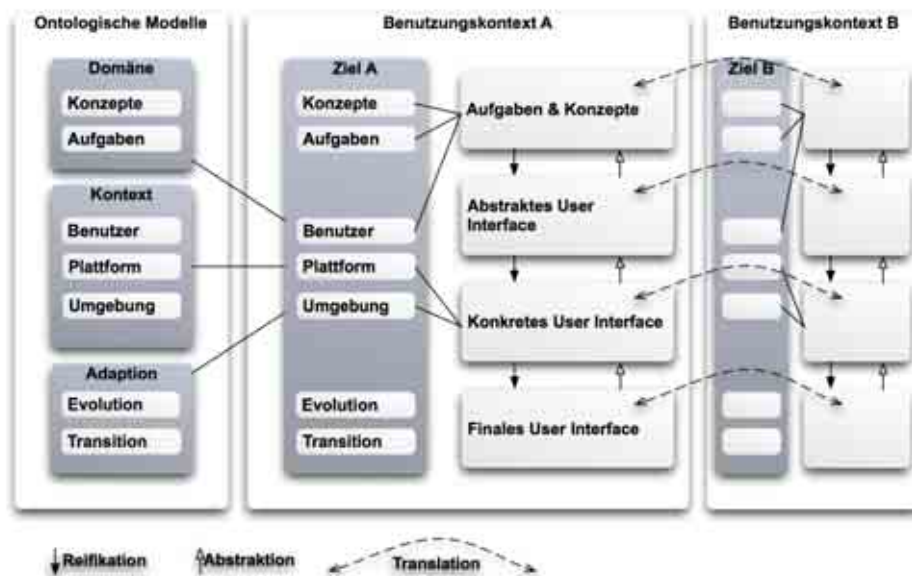


Abbildung 4.10: CAMELEON Referenz Framework nach [CCB⁺03].

Diese Methoden wurden anhand des CAMELEON Referenz Modell konzeptionell integriert [CCB⁺03, CCT⁺02, TCC04]. Das CAMELEON Referenz Modell teilt die Modelle in vier Ebenen auf:

Aufgaben und Konzepte: Auf dieser Ebene werden Aufgabenabläufe (z.B. Ausfüllen eines Fragebogens) spezifiziert, und die involvierten Konzepte (Fragebogen, Teilnehmer, Frage, Antworten, etc.) beschrieben.

Abstraktes User Interface (AUI): Auf dieser Ebene werden abstrakte Gruppierungs- und Interaktionselemente beschrieben, die noch keine Konkrete Darstellung haben.

Konkretes User Interface (CUI): Hier wird auf eine konkrete Darstellung, jedoch noch keinen spezifischen Darstellungstoolkit oder Renderer verwiesen.

Finales User Interface (FUI): Die endgültige Darstellung in einem Spezifischen Format, Toolkit auf eine spezifischen Maschine.

Abbildung 4.10 zeigt, wie die verschiedenen Modellierungsebenen aufeinander aufbauen und im Entwicklungsprozess sequenziell durchlaufen werden. Hierbei herrschen zwei Entwicklungsrichtungen vor. Die Hauptrichtungen werden als *Reifikation*, bei der abstrakte Modelle konkretisiert werden und die *Abstraktion*, bei der konkrete Ebenen wieder abstrahiert werden, bezeichnet. Die Übersetzung auf derselben Ebene wird als *Translation* bezeichnet. Daraus ergeben sich die Entwicklungsrichtungen des *Forward Engineering* bei der die *Reifikation* im Vordergrund steht (s. Abbildung 4.11). Das *Reverse Engineering* ist jener Vorgang bei dem aus einer konkreten Darstellung die abstrakten Modelle herausgezogen werden. Beispiele hierfür sind VAQUITA von Bouillon, *et al.* [BVS02, BVE02, BV02] oder MORE von Gearemync 2003 [GBL03].

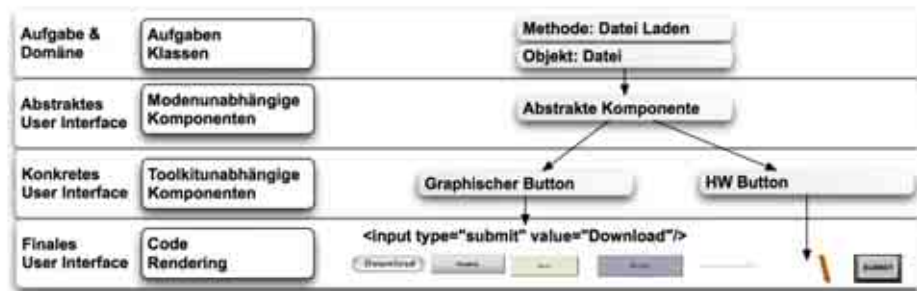


Abbildung 4.11: Reifikationsprozess nach [TCC04].

Generell beschreibt dieses Modell also alle Methoden des *Multi-path Engineering* bei welcher mehrere Entwicklungsstrategien zugleich verfolgt werden können [LVM⁺04, CMP04]. Es ist aus diesem Grund als Referenzmodell plattformübergreifender Entwicklung von Benutzerschnittstellen geeignet.

UsiXML Die User Interface Extensible Markup Language (UsiXML) [LVM⁺04] umfasst eine Familie von Entwicklungswerkzeugen, die von der Arbeitsgruppe um Vanderdonckt im CAMELEON Projekt weiterentwickelt wurden. Diese Entwicklungswerkzeuge bieten Unterstützung bei den verschiedenen Entwicklungsschritten des CAMELEON Modells.

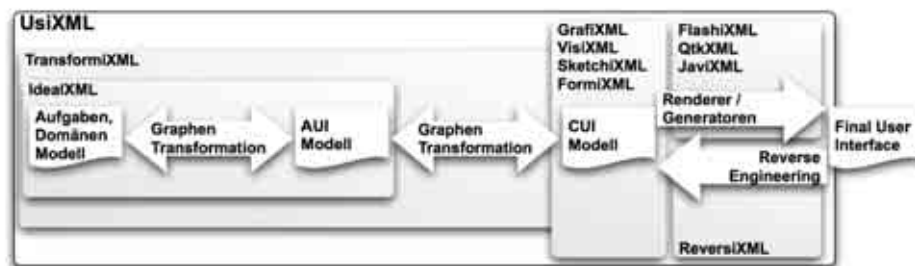


Abbildung 4.12: UsiXML Werkzeuge nach [Van05].

Abbildung zeigt eine Übersicht der Werkzeuge, die in der USiXML Familie zur Verfügung stehen [Van05]. Kernstücke des Entwicklungsprozesses sind die hierbei die *TransformiXML* Plattform, *IdealXML* und zur graphischen Bearbeitung *GrafiXML*. Auf diese Werkzeuge soll im folgenden etwas genauer eingegangen werden.

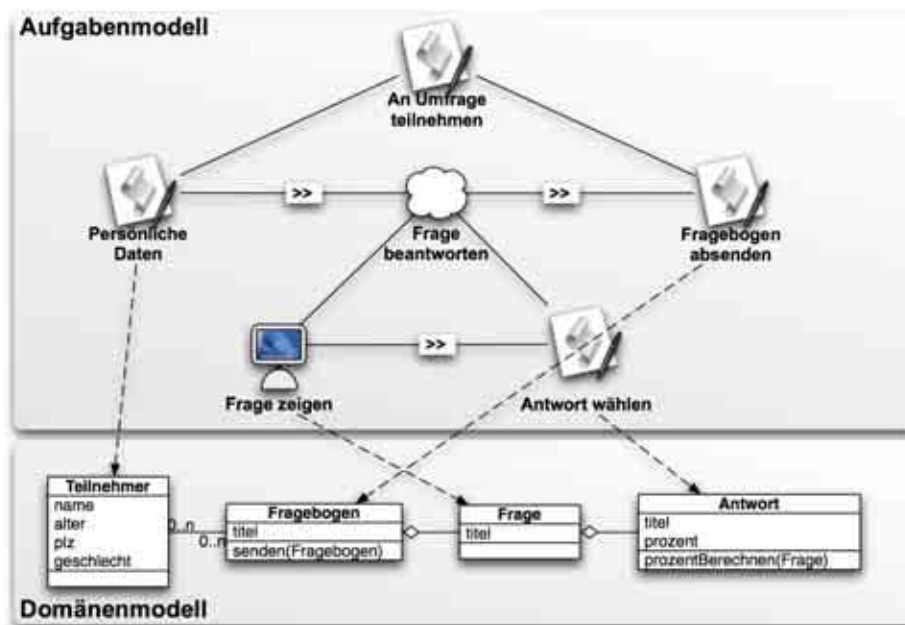


Abbildung 4.13: Abbildung der Aufgaben auf Objekte der Domäne.

TransformiXML ist ein Werkzeug, welches auf der *Attributed Graph Grammars (AGG)* [LV04] nach [TER99] aufbaut. In USiXML können mittels des *IdealXML* Werkzeuges (s. u.) nun Abbildungsregeln zwischen Elementen definiert werden, die durch den *TransformiXML* Parser in AGG Objekte umgesetzt werden. Diese Regeln werden dann auf konkrete USiXML Modelle angewendet und diese so transformiert. Die Beziehungen zwischen Modellen können verschiedene Eigenschaften beschreiben, wie z.B. *Observes*, welche zwischen einem Interaktionsobjekt des AUI und dem Domänenmodell bestehen kann, und ausdrückt, dass das AUI den Wert der Daten darstellt. *Triggers* steht für eine Beziehung zwischen einem AUI Element und einer Methode eines Objektes, im Prinzip eine Abstraktion eines Delegatenkonzeptes. Abbildung 4.13 zeigt das Prinzip der Abbildung von Aufgaben auf Domänenkonzepte.

IdealXML [MSLPGL05] stellt einen graphischen Editor zur Bearbeitung der Aufgaben-, Domänen-, und des AUI-Modells. *IdealXML* ist ein Werkzeug, welches auf Basis von *Design Patterns* agiert und so das Ergebnis der Bearbeitung auch in Form einer leicht modifizierten Alexandrinischen Muster (nach dem Architerken Alexander, dem Erfinder der Design Patterns) in *UsiMXL* und der *Pattern Markup Language (PML)* [SBTZ04] ablegt. *IdealXML* integriert auch das *TransformiXML* Werkzeug um so die Transformationen anstoßen zu können. Abbildung 4.14 zeigt die Oberfläche des *IdealXML* Werkzeuges zur Bearbeitung der Abstrakten Interaktionsobjekte [VB93]. Andere Sichten zeigen einen *Concurrent Task Tree (CTT)* [PMM97] und eine UML Objektsicht der Domänenobjekte.

GrafiXML [LVM⁺04] ist ein graphischer Editor für die Erzeugung von konkreten Benutzerschnittstellen aus den Abstrakten Spezifikationen. Hier kann zum einen ein neues Interface erzeugt werden zum anderen kann eine existierende AUI Beschreibung als Grundlage genutzt werden um ein CUI generieren zu lassen. Abbildung 4.15 zeigt einen Dialog der *GrafiXML* Anwendung.

TERESA Das *Transformation Environment for interactive Systems Representations (TERESA)* von Mori, Paternò, und Santoro [BCMP04, CMP04, MPS04] ist ähnlich wie USiXML an dem CA-

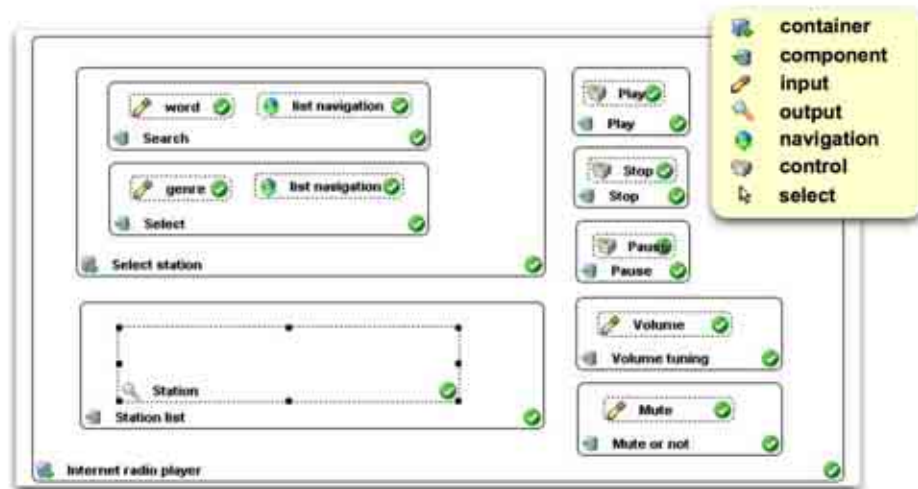


Abbildung 4.14: Gestaltung des AUI mittels des IDEALXML Werkzeuges.

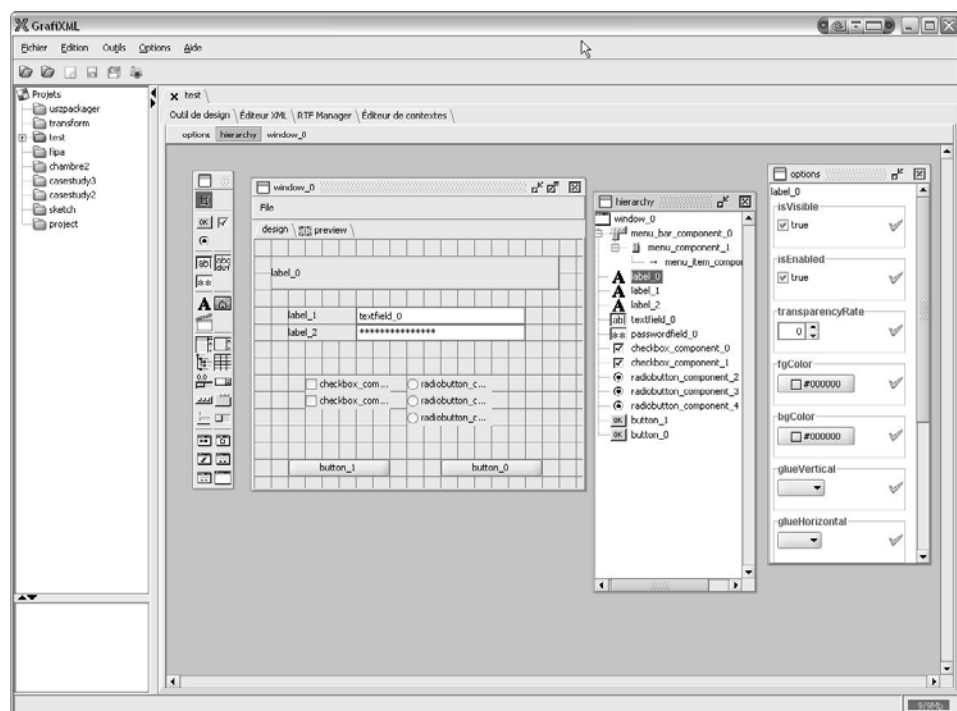


Abbildung 4.15: GRAFIXML Editor.

MELEON Referenzmodell ausgerichtet. Anders als bei USIXML sind allerdings Domänen- und Aufgabenmodell stärker verwoben. Aufbauend auf den *Concurrent Task Trees (CTT)* [CMP04, PMM97, Pat99] werden Aufgaben modelliert. Aufgaben werden dann in Segmente zerlegt, sog. *Presentation Task Sets*. Aus diesen Teilbäumen werden dann die abstrakten Benutzerschnittstellen durch Zuweisung von Interaktoren generiert. In einem weiteren Schritt werden dann die endgültigen Darstellungen in verschiedenen Modalitäten erzeugt.

4.3.3.4 eXtensible Interface Markup Language (XIML)

Mit XIML stellen Puerta und Eisenstein [PE04, PE02] eine Weiterentwicklung früherer Ansätze modellbasierter Entwicklung von Benutzerschnittstellen in der Tradition der Ontologie-Modellierung von Neches [NFF⁺91] vor.

Ausgehend von dem *MASTERMIND* Projekt [SLN93, SSC⁺95] und den dort entwickelten Repräsentationen für Interaktionsdaten entwickelte Puerta mit *MOBI-D* [Pue97, PE98] eine Entwicklungsumgebung, in der Anwendungsoberflächen durch Modelle definiert werden können. Hierzu konnte auch *U-TEL* [TMP98] ein Werkzeug zur Entwicklung von Aufgabenmodellen herangezogen werden. Als Teil dieser Entwicklungsumgebung wurde *MOBILE* [PCOM99] zur graphischen Bearbeitung des Oberflächenmodells entwickelt. In der Arbeit von Puerta und Eisenstein wird die Entwicklung von der reinen modellbasierten Generierung von Anwendungsoberflächen hin zu einer verstärkten Integration von Werkzeugen zur Generierung und Design auf Basis verschiedener Modelle deutlich.

XIML ist eine modellbasierte Beschreibungssprache, die sich auf die Ansätze der Ontologierepräsentation von Neches *et al.* [NFF⁺91] als Grundlage für die Modellierung beruft. Die Syntax selbst verwendet den XML Standard. In seiner ersten Version [PE04] verwendet XIML fünf Modelle in denen die Komponenten der Oberfläche spezifiziert werden: Aufgabe (*task*), Domäne (*domain*), Benutzer (*user*), Dialog und Darstellung (*presentation*). Prinzipiell ist XIML aber offen und kann auch auf weitere Modelle erweitert werden. Zwischen den Elementen, welche in den Modellen definiert werden, können Verknüpfungen angelegt werden und somit das Wissen über die Zusammenhänge encodiert werden. Zusätzlich können über Attribute den Komponenten Eigenschaften zugewiesen werden.

Die Erzeugung von Benutzungsoberflächen in XIML erfolgt auf Basis der drei abstrakten Modelle: Aufgabenmodell, Domänenmodell und Benutzermodell, welche in XML spezifiziert sind. Über Verbindungen (*relations*) zwischen den Komponenten der verschiedenen Modelle, werden die Beziehungen zwischen den Elementen definiert. So kann durch eine Verbindung von Element *D* im Domänenmodell mit der Aufgabe *A* im Aufgabenmodell ausgedrückt werden, dass eine Aufgabe ein bestimmten Sachverhalt beeinflusst. Die Verbindung eines Nutzers mit einer Aufgabe kann die Zugriffserlaubnis dieses Nutzers ausdrücken.

XIML gibt hierbei allerdings keine Vorgaben über die Semantik der Verbindung. Diese zu definieren ist die Aufgabe des Anwendungsentwicklers. Auch die Dialog und Präsentationskomponenten werden über Verbindungen an die zugrundeliegenden Modelle gebunden [EVP01].

Die Erzeugung der Dialog- und Präsentationsmodelle und deren Verknüpfung geschieht halbautomatisch durch sog. *middleware units*. Folgende *middleware units* werden von Puerta und Eisenstein [PE04] beschrieben:

Interactor Selection die im UI darzustellenden Daten und Eingabemöglichkeiten, welche in Domänen und Aufgabenmodell definiert sind müssen im Zuge der UI-Erzeugung mit Interaktionselementen (Dialogmodell) verknüpft werden. Die Auswahl von geeigneten abstrakten Interaktoren wird von dieser *unit* unterstützt.

Presentation Structure Definition Diese *unit* erzeugt zunächst auf Basis des Aufgabenmodells ein Dialogmodell, bindet die (oben) ausgewählten Interaktoren mit den Dialogschritten und verknüpft diese mit den Elementen des Aufgabenmodells. Das eigentlich Darstellungsmodell wird nicht erzeugt sondern in sog. *Mediatoren* ausgelagert.

Contextual Adaptation auf Basis des Plattform-, Benutzer- und des Aufgabenmodells macht diese *unit* Anpassungen der Darstellung, reduziert ihren Umfang oder verändert die Sichtbarkeit von Interaktoren.

Modell	Beschreibung
Aufgabe (Task)	Geschäftsprozesse und Abläufe welche im Interface unterstützt werden, keine Anwendungslogik.
Domäne	Hierarchie von Objekten und Daten ähnlich einer Ontologie
Dialog	Strukturierte Sammlung von Interaktionselementen, welche beschreiben, welche Aktionen dem Benutzer zur Verfügung stehen
Konkrete Interaktion	Hierarchisches Modell konkreter Darstellungselemente
Kontrolle	–
Plattform	Abbildung von Abstrakten Interaktoren auf konkrete Interaktoren und Positionierung entsprechend dem Endgerät.
Umgebung	–
Nutzer	Hierarchisches Nutzermodell, keine kognitive / psychologische Modellierung

Tabelle 4.2: Modellierung in XI ML

Puerta und Eisenstein haben die Umsetzbarkeit des XI ML-Ansatzes anhand einer Beispielanwendung, dem *Map Annotation Assistant (MANNA)* demonstriert. XI ML erscheint aufgrund des Umfangs und der Tiefe der Modellierung als ein sehr ambitionierter Ansatz, der allerdings entsprechend komplex umzusetzen scheint, zumal bislang nur durch wenig Werkzeuge die Arbeit unterstützen.

4.4 Messung der Konsistenz von Benutzerschnittstellen

In Abschnitt 3.3 wurde der Begriff der Konsistenz eingeführt und ein Rahmenmodell zur Einordnung verschiedener Aspekte der Konsistenz von Benutzerschnittstellen nach Kellogg [Kel87, Kel89] beschrieben. Wie Reisner [Rei90] feststellt, ist Konsistenz keine feste Eigenschaft eines Systems oder eines Nutzers. Konsistenz beschreibt die Beziehung zwischen zwei Systemen bezüglich eines oder mehrerer Maße. Eine Benutzerschnittstelle kann also zum Beispiel bezüglich der Anordnung konsistent zu eines Nutzers Erwartung sein. Die Erwartung des Benutzers kann von dessen Domänenwissen über die Aufgaben, welche durch die Anwendung realisiert werden (ökologische Konsistenz), sein Vorwissen aus ähnlichen Anwendungen auf derselben Plattform (Plattformkonsistenz), oder den anderen Dialogen derselben Anwendung (Anwendungskonsistenz) bestimmt sein [DK04]. Die Beschreibung eines generalisierten Referenzmodells plattformübergreifender Konsistenz wird in Abschnitt 5.4 beschrieben.

Es gibt insbesondere im Bereich der Evaluierung von Webseiten eine Anzahl von Werkzeugen, welche automatische Berichte über deren Benutzbarkeit anlegen [BNFV02, BVNF02, OR97]. Viele Systeme werden im Rahmen der Metriken in diesem Abschnitt erwähnt, da diese Systeme meist auf spezifische Aspekte der Gestaltung fokussiert sind. Häufig lassen sich diese Aspekte jedoch nicht unmittelbar auf Kriterien der Benutzbarkeit abbilden, wie es bei der Konsistenz der Fall ist.

Die generelle automatische Evaluierung stößt jedoch auch schnell an ihre Grenzen [FP99] bezüglich der Aussagekraft von Ergebnissen und der Erfassung komplexer Benutzerschnittstellen. Evaluie-

rung muss in einen Entwicklungsprozess eingebunden werden, soll sie für den Entwickler nutzbar gemacht werden. Somit ist die direkte Nutzung von Metriken bei der Gestaltung notwendig. Eine umfassende Übersicht von Evaluierungsmethoden findet sich in [IH01].

Die Dimension, bezüglich derer die Konsistenz eines User Interface definiert werden kann, lässt sich laut Kellogg [Kel87] in die verschiedenen Ebenen der Mensch-Maschine Interaktion [FD82, FDFH90, Nor88] abbilden. Dieses Rahmenmodell wurde in Abschnitt 3.3 beschrieben.

Die Entwicklung von Algorithmen zur Erfassung von User Interface Eigenschaften, sog. User Interface Metriken, welche psychologisch, ästhetisch oder technisch relevante Dimensionen einer Benutzerschnittstelle beschreiben hat bereits eine lange Tradition [Sea95]. Ziel bei der Entwicklung derartiger Maße war es in der Regel, objektive Kriterien bei der Gestaltung von Oberflächen anbieten zu können, welche Entwickler bei der Gestaltung unterstützen können.

Im folgenden Abschnitt wird eine Übersicht über die verfügbaren Metriken gegeben. Deren Anwendbarkeit für die Erfassung der Konsistenz von Benutzerschnittstellen steht hierbei zunächst nicht im Vordergrund. In einem nächsten Schritt werden diese Metriken in das Rahmenmodell von Kellogg [Kel87] eingeordnet und somit einer der Ebenen der Mensch-Maschine Interaktion zugeordnet.

4.4.1 Metriken

Design gilt für viele als intuitive Disziplin in der Erfahrung und Gespür notwendig sind. Dennoch wurde in der Vergangenheit von verschiedener Seite versucht, Designkriterien von Benutzerschnittstellen mittels objektiver Maße zu erfassen und so beschreibbar zu machen.

Da Konsistenz die Beziehung zwischen den Akteuren der Interaktion bzgl. eines oder mehrerer Kriterien beschreibt, ist es notwendig, für die automatische Erfassung, diese Kriterien zu formalisieren. Wo vorhanden, wurden Algorithmen und Methoden beschrieben, welche zur Bewertung des Kriteriums dienen können. Metriken stellen eine Operationalisierung von Gestaltungseigenschaften dar, die in der Regel über Evaluierungsvorschriften definiert werden [Sea95, IH01]. Diese Metriken bieten den Vorteil, Eigenschaften eines Designs *automatisch* evaluieren zu können.

Der automatischen Evaluierung sind jedoch Grenzen gesetzt. Nach Farenc *et al.* [FP99] können nur ein Teil der ergonomischen Vorschriften des Designs operationalisiert werden, da zusätzliche Informationen vom Benutzer, aus dem Kontext oder aus dem funktionalen Kern der Anwendung benötigt werden. Anhand einer Anzahl von 230 Design Regeln evaluierten Farenc *et al.* deren Operationalisierbarkeit und kamen auf einen Anteil von ca. 44% für den Fall, dass keine Informationen aus dem funktionalen Kern der Anwendung verfügbar sind, bzw. 78%, falls diese zur Verfügung stehen.

Die in der Folge beschriebenen Metriken gliedern sich in verschiedene Gestaltungsaspekte graphischer Benutzerschnittstellen, wie Entfernung, Komplexität, Dichte, Gruppierung, Angemessenheit, Interaktion und Ausrichtung. Es wurde versucht, die wesentlichen Ansätze zusammenzufassen und zum Teil in formalisierter Form wiederzugeben. Bemerkenswert ist, dass die direkte Messung der Konsistenz in keinem Fall beschrieben wurde, sondern unter Verwendung der Metriken über die Beziehung zwischen Elementen.

4.4.1.1 Entfernung

Die Entfernung zweier Elemente in einem Dialog spielt insbesondere dann eine große Rolle, wenn eine inhaltliche Beziehung durch graphische Mittel ausgedrückt werden soll. In den Ordnungsprinzipien der *Gestalt-Psychologen* spielt die Nähe eine zentrale Rolle. Kothari und Basak [KB02] weisen darauf hin, dass die wahrgenommene Distanz sich in manchen Fällen deutlich von den rechnerisch erfassten Distanzen unterscheiden.

Der gängigste Ansatz zur Berechnung der Distanz basiert auf der *Euklidischen Distanz* zwischen den Mittelpunkten der Elemente. Handelt es sich bei einem Element um ein zusammengesetztes Objekt oder um eine Form, welche nicht rechteckig ist, wird diese meist durch ein Rechteck, die sog. *Bounding Box* approximiert. Wie Kothari und Basak [KB02] diskutieren, berücksichtigt dies jedoch

nicht Eigenschaften der beiden Elemente wie deren (relative) Größe, oder den Abstand zwischen den Flächen der beiden Elemente.

Ein alternatives Maß stellt die *Hausdorff'sche Distanz* dar [Rot91]. Dieses gleicht zwar bezüglich zweier Mittelpunkte der Euklidischen Distanz, betrachtet man die Ecken der Bounding Box eines oder mehrerer Elemente jedoch als eine Punktwolke, kann die Hausdorff'sche Distanz jedoch genutzt werden, um die Distanz zwischen zwei Elementen als Entfernung zwischen deren einzelnen Eckpunkten zu bestimmen.

$$\hat{h}(A, B) = \max_{a \in A} \{ \min_{b \in B} \{ d(a, b) \} \} \quad (4.1)$$

Zur Berechnung der einzelnen Distanzen zwischen den Punkten des Elements A ($\{a_1, a_2, \dots, a_n\}$) als $d(a, b)$ wird in der Regel die Euklidische Distanz verwendet. Für die Berechnung der Hausdorff'schen Distanz wird nun für jedes a zunächst das Minimum aller Distanzen zu einem der Punkte b ermittelt. Aus diesen wird nun wiederum die größte der Distanzen als Hausdorff'sche Distanz verwendet. Da die so ermittelte Distanz gerichtet ist, muss als Generalisierung die größere beider gerichteter Distanzen gewertet werden.

Die Hausdorff'sche Distanz hat den Nachteil, dass sie bei symmetrischen Objekten, wie bei vertikal versetzten Elementen nicht sensitiv für die Unterschiede in der tatsächlich wahrgenommenen Distanz ist. Basierend auf diesen Überlegungen schlagen Kothari und Basak [KB02] ein Distanzmaß vor, welches die perzeptuellen Eigenschaften der Entfernung zwischen Elementen berücksichtigen soll. Hierzu wird zunächst die *Steigung der perzeptuellen Verbindungsachse* (θ) der Elemente bestimmt. Die Ausrichtung dieser Achse bestimmt sich daraus, wie sich die Flächenanteile (q_{ij}) des zweiten Elementes über die umgebenden Quadranten (ausgerichtet an den Kanten der Bounding Box; beginnend rechts) des ersten Elementes verteilen.

$$\theta_{ij} = \sum_{k=1}^8 \left(\frac{\pi}{4} \right) \rho_{ijk} (k-1) \quad (4.2)$$

$$\rho_{ij} = \frac{q_{ij}}{\sum_{k=1}^8 q_k^8} \quad (4.3)$$

In einem zweiten Schritt wird dann die Nähe der Elemente als *Variante der Hausdorff'schen Distanz* der Projektionen der Punkte auf die Gerade mit der Steigung θ berechnet. Der Effekt der Nähe, d. h. dass Differenzen im Nahbereich stärker wahrgenommen werden, wird durch den Faktor f berücksichtigt.

$$p_{ij} = \sum_{a \in A} f(\min_{b \in B} d(a', b')) + \sum_{b \in B} f(\min_{a \in A} d(a', b')) \quad (4.4)$$

$$f(x) = \frac{1}{1 + (x/\lambda)^2} \quad (4.5)$$

Für λ schlagen Kothari und Basak [KB02] Werte um 25 vor. Weiterhin wird eine Korrektur bezüglich der Dominanz horizontaler und vertikaler Ausrichtung gegenüber diagonaler Ausrichtung vorgeschlagen.

Das Distanzmaß von Kothari und Basak berücksichtigt nicht nur einen Punkt eines Elementes sondern die Verteilung der Punkte eines Elements entlang der Verbindungsachse der relativen Schwerpunkte. Die empirische Überprüfung ihrer Hypothese, dass dieses Maß die Eigenschaften menschlicher Wahrnehmung stärker berücksichtigt als konventionelle Distanzmaße bleiben Kothari und Basak trotz augenscheinlicher Validität des Ansatzes zunächst schuldig, ebenso wie die Motivation sowie genauere Justierung des λ Faktors.

4.4.1.2 Komplexität

Wandmacher [Wan03a] definiert Komplexität als „Unsicherheit oder Komplement der Vorhersagbarkeit der räumlichen Positionen der verschiedenen Einheiten.“ Hierbei unterscheidet er in *apriorische Komplexität* als Unsicherheit der räumlichen Positionen der verschiedenen Einheiten *vor* Darbietung der Bildschirmseite und *aposteriorische Komplexität* als Unsicherheit der räumlichen Positionen bei der visuellen Verarbeitung der Bildschirmseite, also *nach* Darbietung der Bildschirmseite.

Die apriorische Komplexität ist bestimmt durch die Menge der Einheiten. Deren räumliche Organisation durch funktionale Gruppierung. Die apriorische Komplexität bestimmt sich als informationstheoretisches Maß der Menge und Komplexität der Information, welche durch das Interface dargestellt wird. Die aposteriorische Komplexität hingegen ist in erster Linie ein Aspekt der Wahrnehmung und bestimmt die Verarbeitung der Information. Perzeptuelle Gruppierung, Unterscheidbarkeit, Erkennbarkeit, Aufmerksamkeitslenkung, etc., sowie die konsistente räumliche Anordnung bestimmen die Komplexität.

Comber und Maltby [CM94] schlagen eine Formel zur Bestimmung der *Layout Complexity* (entspricht der aposteriorischen Komplexität) vor, welche auf den typographischen Regel von Bonsiepe [Bon68] aufbaut.

$$C = -N \sum p_i \log_2 p_i \quad (4.6)$$

$$N = \sum n_i \quad (4.7)$$

$$p_i = \frac{n_i}{n} \quad (4.8)$$

Die Komplexität (C) berechnet sich hierbei aus der Gesamtzahl der Elemente (N), der Anzahl der Klassen (n ; d. h. Elementgruppen mit gleicher Breite, Höhe oder Abstand), der Anzahl der Elemente in der i -ten Klasse (n_i) und der relativen Größe der i -ten Klasse (p_i). *Layout Complexity* ist somit ein quantitatives Maß für die relative Ordnung von Interface Elementen. Es wurde in verschiedenen Studien auf seine Validität hin überprüft, wie z. B. Textlayout oder textbasierten Oberflächen. Auch für die Evaluierung von graphischen Benutzungsoberflächen wurde *Layout Complexity* als Maß verwendet [CM94]. In einer Untersuchung [CM94, CM96] zeigen Comber und Maltby dass eine mittlere Komplexität zu einer optimalen Benutzbarkeit führt.

Ngo *et al.* [NSA00, NB01] stellen eine ganze Reihe ästhetisch motivierter Metriken vor, die hier zu einem großen Teil auch aufgeführt wurden. Komplexität ist im Sinne von Ngo *et al.* die gewichtete Summe all dieser Metriken zu einem Gesamtwert als Maß für die Ordnung des Dialoges.

4.4.1.3 Dichte

Dichte ist ein Kriterium der Informationsmenge einer Oberfläche. Die Dichte stellt die Menge der Information dar, welche auf einer bestimmten Fläche eines Dialoges enthalten ist. Tullis [Tul83, Tul88b] entwickelte ein Werkzeug zur Beurteilung von alphanumerischen Benutzerschnittstellen. Zwei der erfassten Kriterien betrafen die Dichte des Dialoges. Zum einen erfasste Tullis die *Zeichendichte* als Anteil der Oberfläche, die zur Darstellung von Information verbraucht wird. Die lokale Zeichendichte wurde als Anzahl der Zeichen im Bereich eines 5°Sichtwinkels (entspricht 15 Zeichen und 7 Zeilen) berechnet. Ngo *et al.* [NSA00] übertrugen das Maß von Tullis auf graphische Oberflächen, indem die Dichte (D) als Anteil der Fläche der Objekte (A_i) an der Gesamtfläche (A) definierten.

$$D = 1 - \frac{\sum A_i}{A} \quad (4.9)$$

4.4.1.4 Gruppierung

Die Gruppierung von Elementen ermöglicht es, einem Dialog eine innere Gliederung zu geben. Gruppierungen erfolgen in der Regel in einem perzeptuellen Sinne, d. h. mit Hilfe von Gestaltungstechniken werden Elemente zu Wahrnehmungseinheiten zusammengefasst. Basierend auf den Gesetzen der Organisation der Gestalt-Psychologen aus dem frühen zwanzigsten Jahrhundert [Wer12, Gol89] lassen sich folgende Prinzipien der perzeptuellen Organisation von Elementen eines Dialoges zusammenfassen. Diese Eigenschaften sind so grundlegend, dass sie aufgeführt werden, auch wenn sie nicht alle unmittelbar in Metriken überführbar sind bzw. überführt wurden.

Gesetz der Nähe Objekte die in unmittelbarer räumlicher Nähe zueinander stehen werden leicht als Gruppe wahrgenommen.

Gesetz der guten Form (Prägnanz) oder Gesetz der Einfachheit Eine Anordnung von Objekten wird stets so wahrgenommen werden, dass sich eine komplexe Anordnung in einfache Grundformen zerlegt.

Gesetz der Ähnlichkeit Ähnliche Objekte werden leichter zu Gruppen zusammengefasst wahrgenommen als unähnliche Objekte.

Gesetz der guten Fortsetzung Eine Folge von Objekten die eine Linie bilden wird von einem Betrachter auch über Störungen oder Überdeckungen hinweg fortgesetzt.

Gesetz der Geschlossenheit Stellt eine Reihe von Objekten eine einfache Form dar, werden kleine Unterbrechungen in der Form in der Wahrnehmung des Betrachters ‚repariert‘. Die Form wird geschlossen.

Gesetz des gemeinsamen Schicksals Bewegen sich zwei Objekte in dieselbe Richtung werden sie als zusammengehörig wahrgenommen.

Gesetz der Vertrautheit erscheinen Gruppen von Objekten vertraut oder ergeben diese Gruppierungen einen inhaltlichen Sinn werden diese Objekte leichter zu einer Gruppe zusammengefasst.

Gesetz der Eingeschlossenheit oder der gemeinsamen Fläche Objekte die in derselben Fläche liegen, bzw. von der selben Form eingeschlossen werden, werden als Gruppirt wahrgenommen.

Gesetz der uniformen Verknüpfung Objekte die durch eine Linie oder die direkte Angrenzung verknüpft sind werden als Gruppe wahrgenommen [PR94].

Gesetz der Orientierung Objekte die dieselbe Orientierung besitzen (horizontal / vertikal) werden als zusammengehörig empfunden [Ols70].

Gruppierungen können durch verschiedenen Eigenschaften beschrieben werden, wie die Umrissform der Gruppe, d. h. die Form, welche sich aus der Gruppierung der Elemente ergibt, die Hauptachse und der Schwerpunkt der Gruppe oder die Ordnung bzw. das Muster der Gruppe.

Innerhalb einer Gruppe, bzw. zwischen Gruppen, können wiederum Gruppierungseffekte und andere Effekte auftreten, wie z. B. die 1915 von Rubin beschriebenen Effekte der *Figur - Grund - Unterscheidung* [Rub15]. Diese beschreibt, wie sich Formen aus einem Muster abheben und sich perzeptuell vom Hintergrund abtrennen. Manche Eigenschaften des Verhältnisses zwischen Formen scheinen die Wahrnehmung einer Form als Figur gegenüber der anderen Form als Hintergrund zu unterstützen. Zu den Eigenschaften, welche helfen, eine Figur zu bilden, gehört die *Symmetrie*, die *Konvexität*, eine verhältnismäßig *kleinere Fläche*, vertikale oder horizontale *Orientierung* und eine *bedeutungstragende Einheit*.

Tullis [Tul83, Tul88b] verwendete bei der Erfassung von Eigenschaften textbasierter Interfaces auch Gruppenmaße wie die *absolute Anzahl* (N) der Gruppierungen (G) und die *mittlere Gruppengröße* (\bar{G}) als Mittelwert der Flächengröße (S) der Gruppierung multipliziert mit der Anzahl der Einheiten ($n(Z)$) in einer Gruppierung.

$$N = \sum G \quad (4.10)$$

$$\bar{G} = \frac{\sum S_i \times n(Z)}{N} \quad (4.11)$$

Ngo *et al.*[NSA00] stellen *Kohäsion* als Maß für die Übereinstimmung der relativen Höhen- und Breitenmaße der einzelnen Objekte vor. Dieses Maß kann damit einen Aspekt des Gesetzes der Ähnlichkeit abbilden und den perzeptuellen Zusammenhalt einer Gruppe messen.

$$C = \frac{|C_{fl}| + |C_{lo}|}{2} \quad (4.12)$$

$$C_{fl} = \begin{cases} t_{fl} & (ift_{fl} \leq 1) \\ 1/t_{fl} & (otherwise) \end{cases} \quad (4.13)$$

$$C_{lo} = \frac{\sum_i^n f_i}{n} \quad (4.14)$$

$$t_{fl} = \frac{h_{layout}/b_{layout}}{h_{frame}/b_{frame}} \quad (4.15)$$

$$f_i = \begin{cases} t_i & (ift_i \leq 1) \\ 1/t_i & (otherwise) \end{cases} \quad (4.16)$$

$$t_i = \frac{h_i/b_i}{h_{layout}/b_{layout}} \quad (4.17)$$

Wobei *Kohäsion* hier aus der Kohäsion (C_{fl}) des Seitenverhältnisses des Layouts (h_{layout}/b_{layout} ; z.B. des Gruppenumrisses) mit dem Seitenverhältnis des Dialograhmens (h_{frame}/b_{frame}) und der Kohäsion (C_{lo}) des Seitenverhältnisses der einzelnen Elemente (h_i/b_i) wiederum mit dem Seitenverhältnis des Layouts besteht. Beide Maße können allerdings ebenso getrennt verwendet werden.

Als ein zweites Kohäsionsmaß mit Bezug auf die anderen Aspekte der einzelnen Objekte führen Ngo *et al.*[NSA00] ein Maß für die *Einheit* ein. Dies beschreibt die Gesamtheit der Elemente, die visuell eine Einheit bilden. Den Gestalt-Prinzipien der Ähnlichkeit und der Nähe folgend, berechnet sich Einheit demzufolge als:

$$U = \frac{|U_{form}| + |U_{space}|}{2} \quad (4.18)$$

$$U_{form} = 1 - \frac{n_{size} + n_{color} + n_{shape} - 3}{3n} \quad (4.19)$$

$$U_{space} = 1 - \frac{a_{layout} - \sum_i^n a_i}{a_{frame} - \sum_i^n a_i} \quad (4.20)$$

Die Einheit der Form (U_{form}) wird über die Anzahl der verschiedenen Größen, Farben und Formen ($n_{size}, n_{color}, n_{shape}$) im Dialog beschrieben. Die Einheit des Raumes (U_{space}) hingegen als Verhältnis der Flächenanteile der einzelnen Elemente (a_i) am Layout (a_{layout}) gegenüber der Anteile der Elemente an der Gesamtfläche (a_{frame}) beschrieben wird.

4.4.1.5 Angemessenheit des Layouts

Angemessenheit des Layouts (*Layout Appropriateness*) wurde von Sears [Sea93] geprägt. Es handelt sich um ein Maß, welches auf *Fitts' Law* beruht. Dieses besagt, dass die Zeit die man benötigt um mit dem Mauszeiger ein Element zu erreichen, proportional zu dem Logarithmus aus dem Quotienten von Entfernung und Größe des Elements ist [Fit54]. Die *Layout Appropriateness* ist um so besser, je kürzer die Wege sind, die der Benutzer beim erledigen einer Aufgabe im Interface zurücklegen muss.

Dazu werden entweder empirisch oder induktiv die Häufigkeiten der Transitionen zwischen allen Elementen im Interface festgelegt. Um eine optimale *Layout Appropriateness* zu erreichen, muss nun der insgesamt zurückgelegte Weg reduziert werden, die Elemente also im Pfad der Interaktion möglichst auf einer Linie, bzw. ohne Umwege angeordnet sein. In der Regel legt Sears die Interaktionsrichtung von links oben nach rechts unten fest. Durch die Nutzung einfacher Transitionsnetzwerke spart sich Sears die aufwändige *GOMS* Analyse die bei anderen Ansätzen, wie dem System *USAGE* von Byrne *et al.* [BWS⁺94]. Dieses nutzt die *GOMS* Methode um die Effizienz eines Interfaces zu messen. Das Design wird aus einer formalen Beschreibung generiert. Auf der formalen Beschreibung führt *USAGE* eine Analyse durch, welche die Aktionen in der Oberfläche extrahiert. Auf der extrahierten Beschreibung kann eine *GOMS* Analyse durchgeführt werden.

Bei Sears berechnet sich der Interaktionsaufwand als Grundlage für die Berechnung der Angemessenheit als:

$$cost = \sum FrequencyOfTransition \times CostOfTransition \quad (4.21)$$

Zur Berechnung der Einzelaufwände kann eine beliebige Formel verwendet werden. Sears schlägt hier *Fitts' Law* vor.

Sears [Sea95] stellte die Umgebung *AIDE* zur Evaluation von Interfaces zur Designzeit als Unterstützung des Designers durch verschiedene Metriken vor. Diese können verschieden gewichtet in den Design-Prozess einfließen und entweder bewertend als Information dem Entwickler zur Verfügung gestellt werden oder als Grundlage eines automatischen Layouts dienen. *AIDE* wendet hierbei einen Optimierungsalgorithmus an, um die optimale Position der Elemente im Dialog zu bestimmen. Eine der Metriken die hier eingesetzt wurden, war die Effizienz als Entfernung, die der Benutzer mit der Maus zurücklegen muss. Dieses Maß stützt sich auf frühere Arbeiten Sears in denen er die Angemessenheit von Layout (*Layout Appropriateness*) gemessen hatte [Sea93].

4.4.1.6 Interaktionspunkte

Rauterberg [Rau96] führt das Konzept der Interaktionspunkte (*Interaction Points*) ein und nutzt es um verschiedenen Interaktionskonzepte zu klassifizieren. Er betrachtet die Benutzerschnittstelle als Interaktionsraum, der aus zwei überlappenden Räumen besteht: dem Objektraum (Elemente der Oberfläche) und dem Funktionsraum (verfügbare Funktionen). Sowohl Objekte als auch Funktionen müssen unterschieden werden in sichtbare und versteckte Elemente.

Ein interaktives System kann unterschieden werden in zwei Bereiche. Der eine Bereich wird von einem Dialogmanager verwaltet, der andere von der Anwendung selbst. Folglich müssen auch Objekte und Funktionen unterschieden werden, ob sie zum Dialogmanagement (Fenster schließen) oder zur Anwendung gehören (Dokument speichern). Rauterberg stellt außerdem noch die Maße der Anwendungs- und Dialogflexibilität vor [Rau93, Rau95].

Jede Funktion kann einen oder mehrere Interaktionspunkte haben, über welche die Funktion aufgerufen werden kann (also Interaktionspunkt der Dialogfunktion versus Interaktionspunkt der Anwendungsfunktion). Diese Interaktionspunkte werden als Oberflächenelemente realisiert. Es gibt also sichtbare und versteckte Interaktionspunkte für Funktionen.

Rauterberg leitet hieraus die Maße des *funktionalen Feedbacks* und der *interaktiven Direktheit* ab. *Funktionales Feedback* definiert sich dadurch, wie viele Funktionen durch sichtbare Interaktionspunkte zugänglich sind. Das bedeutet wie viele Funktionen kann ich sehen?

$$FunctionalFeedback = \frac{1}{D} \sum \left(\frac{\#PF_D}{\#HF_D} \right) \times 100\% \quad (4.22)$$

Wobei D die Anzahl der verschiedenen Dialog Kontexte beschreibt. $\#PF$ steht für die Anzahl der sichtbaren funktionalen Interaktionspunkte (Anwendung und Dialog) und $\#HF$ für die Anzahl der versteckten funktionalen Interaktionspunkte. Die Anzahl der verschiedenen Dialog-Pfade wird mit der Variablen P beschrieben. Die Länge des Pfades $l(Path)$ steht für die Strecke, die durchlaufen werden muss, um einen Interaktionspunkt zu erreichen.

Interaktive Direktheit definiert sich dadurch, wie viele Funktions-Interaktionspunkte direkt aufrufbar sind. Also, wie schnell kann ich auf Funktionen zugreifen?

4.4.1.7 Ausrichtung

Die Ausrichtung der Elemente eines Dialoges beschreibt die Anordnung der Elemente relativ zu den Bezugspunkten des Dialoges. Man unterscheidet folgende Aspekte der Ausrichtung.

- Balance
- Gleichgewicht
- Symmetrie

Die *Balance* stellt die Verteilung des optischen Gewichtes im Dialog dar. Das Gewicht eines Objektes wird durch verschiedene Eigenschaften bestimmt. So wirken dunkle Objekte schwerer, ebenso wie große Objekte oder ungewöhnliche Formen. Balance betrifft sowohl die horizontale als auch die vertikale Verteilung der Gewichtung. Das Maß der Balance wird in den meisten Arbeiten zur Evaluierung der Gestaltung der Oberfläche verwendet [KF93, Sea95, SW84, VG94]. Die Berechnung der Balance von Ngo *et al.* [NSA00, NB01] allerdings, gehört zu den aufwändigsten Ansätzen. Ngo *et al.* [NSA00] beschreiben eine Formel zur Berechnung der Balance (B) auf Basis der Größe (a), Farbe (c) und Form (s) eines Objektes.

$$B = 1 - \frac{|B_{vertical}| + |B_{horizontal}|}{2} \quad (4.23)$$

$$B_{vertical} = \frac{w_L - w_R}{\max(|w_L|, |r_R|)} \quad (4.24)$$

$$B_{horizontal} = \frac{w_T - w_B}{\max(|w_T|, |r_B|)} \quad (4.25)$$

$$w_j = \sum_i^{n_j} d_{ij} \left(\frac{a_{ij}}{a_{max}} + |c_{ij} - c_{frame}| + s_{ij} \right); j = L, R, T, B \quad (4.26)$$

Diese Formel berechnet die horizontale und vertikale Balance ($B_{horizontal}$, $B_{vertical}$) auf Basis der Summe der Abweichungen von den Mittelachsen des Dialoges (d_{ij}). Diese Abweichung wird gewichtet durch die relative Größe des Objektes und den Farbkontrast des Objektes vom Hintergrund und die Form. Zur Beschreibung der Farbe wird ein kontinuierlicher Wert zwischen 1 (Schwarz) und 0 (Weiß) eingesetzt. Die Form wird mittels eines Maßes zur Beschreibung von Formen nach Birkhoff [Bir33] festgelegt. Die Buchstaben L und R , bzw. T und B stehen für links und rechts, bzw. oben und unten.

Das *Gleichgewicht* der Darstellung wird von Ngo *et al.* [NSA00] als Übereinstimmung des Schwerpunktes des Layouts mit dem Mittelpunkt des Dialoges definiert. Ngo *et al.* schlagen eine Formel zur Berechnung des Gleichgewichts vor, auf welche an dieser Stelle nicht näher eingegangen werden soll.

Die *Symmetrie* eines Dialoges wird durch die Übereinstimmung des Layouts bezüglich einer axialen oder Punkt-Spiegelung bestimmt. Lässt sich ein Dialog über die Längs- oder Querachse bzw. radial über einen Punkt spiegeln, dann ist er bezüglich der jeweiligen Achse symmetrisch [VG94, KF93]. Kim und Foley [KF93] betrachten Symmetrie bezüglich der horizontalen und der vertikalen Achse. Ngo *et al.* [NSA00] schlagen eine Formel zur Berechnung der Symmetrie über alle drei Achsen vor

bei der die Beträge der Überlagerungen der Elemente in den entsprechenden Quadranten berechnet werden.

Ein Werkzeug zur Erfassung verschiedener Aspekte der Oberflächengestaltung, und hierbei insbesondere der Anordnung wurde von Kim und Foley entwickelt. Kim und Foley [KF93] bezogen sich hierzu auf einen Satz von sieben vermutlich aufgabenunabhängigen Maßen, wozu gehörten: Links-Rechts- und Oben-Unten-Relation, vertikale und horizontale Symmetrie. Andere Maße wie Höhe-Breite-Relation, Anteil ungenutzten Raumes und Größenabweichungen wurden bereits in anderen Ansätzen beschrieben. Ein weiterer Ansatz, der den Schwerpunkt auf die Anordnung der Elemente in der Oberfläche legt ist der von Streveler und Wassermann [SW84].

4.4.1.8 Einordnung

Der Hintergrund dieser Zusammenfassung existierender Metriken zur Operationalisierung und Evaluierung der Gestaltung von Benutzerschnittstellen, war der Gedanke, dass zur automatischen Erfassung der Konsistenz relevante Metriken zur Erfassung der Aspekte der Konsistenz verfügbar sein müssen. Trifft die These Reisners [Rei90] zu, dass Konsistenz ein relationales Maß zwischen verschiedenen Aspekten des Benutzers und der Anwendung ist, müssen zunächst Methoden entwickelt werden, um diese Aspekte zu messen. Der Bereich der Beschreibung verschiedener Benutzertypen [KF88] wurde in dieser Arbeit bewusst ausgelassen und der Fokus auf die Aspekte der Gestaltung gelegt.

Ziel dieses Abschnittes war es also, Metriken für die Evaluierung von Benutzerschnittstellen zu sammeln. In Abschnitt 3.3 wurde der Begriff der Konsistenz beschrieben und das System zur Beschreibung der Dimensionen der Konsistenz von Kellogg [Kel87] präsentiert. Dieses Schema wird nun in Bezug gesetzt mit den hier beschriebenen Metriken, um zu zeigen, welche der Metriken für die Operationalisierung welcher Dimension der Konsistenz geeignet scheint. Tabelle 4.3 zeigt eine Zusammenfassung in Anlehnung an Tabelle 3.1.

		Metriken
Konzeptuell	Aufgaben-Ebene	Layout Appropriateness [Sea93]
	Semantische Ebene	
Kommunikation	Syntaktische Ebene	Gestaltgesetze [Wer12, PR94, Ols70], Mittlere Gruppengröße [Tul83], Kohäsion [NSA00], Einheit [NSA00]
	Interaktionsebene	Interaktionspunkte [Rau96], Komplexität, Layout Complexity [CM94]
Physikalisch	Räumliches Layout	Euklidische Distanz, Hausdorff'sche Distanz, Perzeptuelle Verbindung [KB02], Gleichgewicht, Balance, Symmetrie, Dichte [NSA00]
	Geräte-Ebene	

Tabelle 4.3: Einordnung der Metriken in Kelloggs Schema zur Definition von Konsistenz.

Wie sich zeigt, sind fast alle Dimensionen der Konsistenz nach Kellogg durch eine der hier aufgeführten Metriken abgebildet. Selbst die aus Sicht der Oberfläche ferne und abstrakte Aufgaben-Ebene, auf der die Passung zwischen Aufgabe und Nutzervorstellung beschrieben wird ist (zwar nur partiell) durch die Layout Angemessenheit nach Sears messbar. Die Messung der semantischen Ebene ist, aufgrund der problematischen Operationalisierung der Semantik schwierig und in keiner der Metriken vertreten.

Die auf der syntaktischen Ebene abgebildeten Organisationsprinzipien werden durch mehrere Maße, wie die Gruppengröße, Kohäsion und Einheit, sowie die Gestaltgesetze abgebildet. Interaktionspunkte und der apriorischen Komplexität adressieren direkt die Probleme der konsistenten Gestaltung der Interaktion. Für das räumliche Layout ist die direkte Übersetzung in Maße am einfachsten

zu erreichen, daher können hier beliebig viele Maße identifiziert werden, wie die Anordnungsmaße, Dichte und Entfernung.

Die Metriken zur Erfassung der Oberflächeneigenschaften sind somit vorhanden und können nun als Eingabe für ein System zur Berechnung und Evaluierung der Konsistenz eingesetzt werden. Dieses wird in Abschnitt 5.4 ausführlich beschrieben werden.

4.4.2 Messung der Konsistenz

Trotz der Bedeutung der Konsistenz für die Gestaltung benutzbarer Oberflächen, hat es in der Vergangenheit nur sehr wenige Ansätze zur Entwicklung von Werkzeugen gegeben. Der überwiegende Teil der Arbeiten konzentriert sich auf die implizite Berücksichtigung der Konsistenz als Resultat maschinengenerierter Benutzerschnittstellen durch UIMS (s. Abschnitt 4.3).

Im vorangegangenen Abschnitt wurden verschiedene Maße zur Evaluierung von Benutzerschnittstellen beschrieben, welche zum Teil auch im Rahmen von entwicklungsunterstützenden Werkzeugen eingesetzt wurden. An der gegebenen Stelle wurde jeweils auf die betreffenden Werkzeuge verwiesen. Keines dieser Werkzeuge adressiert jedoch ausdrücklich die Konsistenz einer Anwendung. In den meisten Fällen stehen gestalterische Aspekte [VG94] im Vordergrund.

Wahrnehmungspsychologisch relevante Eigenschaften wie Komplexität [CM94] oder Entfernung [KB02], oder leistungsrelevante Kriterien wie Angemessenheit [Sea93] werden hier betrachtet ohne direkt in den Kontext der Konsistenz gesetzt zu werden. Gleichwohl ist es möglich, dass die Berücksichtigung einiger dieser Maße bei der Messung der Konsistenz eine Rolle spielen.

Zu den wenigen Werkzeugen, welche explizit das Thema der Konsistenzmessung adressieren, gehört SHERLOCK von Mahajan und Shneiderman [MS97, Mah96, SCJS95, MS95]. Bei SHERLOCK handelt es sich um eine Familie von Werkzeugen zur Messung verschiedener Aspekte der Konsistenz welche modular aufgebaut ist. SHERLOCK dient in erster Linie dazu, Dialoge zu bewerten und die Ergebnisse an den Designer zurück zu melden. Die Ergebnisse der Evaluierung werden in Ergebnis-Dialogen zusammengefasst. Konzepte zur direkt-manipulativen interaktiven Entwicklungsunterstützung werden nicht angeboten.

SHERLOCK arbeitet auf kanonischen textuellen Beschreibungen der Dialoge, in denen die verschiedenen Eigenschaften, wie Relationen (*parent*), Anordnung und Größe (*left*, *right*, *width*, *height*), Typographie (*font*), Farben (*foreground*, *background*) und textuellen Inhalten (*label*). Ein Satz an verschiedenen Werkzeugen wurde entwickelt, um anhand dieser Maße die Konsistenz der Oberfläche zu erfassen:

Dialog Box Summary fasst die visuellen Eigenschaften der Dialoge zusammen. Für jeden Dialog der Anwendung werden in einer Tabelle die Werte für die entsprechenden Metriken beschrieben. Die hier verwendeten Metriken stellen eine Untermenge von ca. 40 Metriken dar, welche in der Entwurfsphase als möglicherweise relevant beurteilt wurden:

- Seitenverhältnis (*Aspect Ratio*): Relation Höhe zu Breite, Werte von 0.5 bis 0.8 sind wünschenswert.
- Freie Fläche (*Non-Widget Area*): Relation der Fläche die nicht von Elementen belegt ist zur Gesamtfläche, niedrige Werte zeigen hohe Benutzung an.
- Dichte bzw. Überfülltheit (*Widget Density*): Relation von Elementen auf oberster Ebene zu Gesamtfläche.
- Ränder (*Margins*): Anstand der äußeren Elemente vom Rand, diese sollten für alle Richtungen ähnlich sein.
- Rasterung (*Gridedness*): die Anzahl der verschiedenen X und Y Positionen der Elemente in einem Dialog. Je stärker das Layout einem Raster folgt, desto geringer ist dieser Wert.
- Balance (*Area Balances*): Relation der belegten Fläche auf der rechten versus linken, bzw. oberen versus unteren Hälfte des Dialogs.
- Abweichungen des Schriftsatzes (*Distinct Typefaces*).
- Abweichungen der Hintergrund- bzw. Vordergrundfarben (*Distinct Background / Foreground Colors*).

Margin Analyzer wertet die Abstände aller äußeren Elemente vom Dialogrand aus und bestimmt die häufigsten Werte als die angenommenen Optima.

Concordance Tool sammelt die textuellen Inhalte, insbesondere in der Kurzbeschreibungen (*labels*) und sortiert diese nach Ähnlichkeit. Der Sinn hiervon ist es, leichte Variationen in der Benennung aufzuzeigen und den Entwickler bei der konsistenten Benutzung von Beschriftungen zu unterstützen.

Interface Concordance Tool sammelt ebenfalls alle Beschriftungen aus den Dialogen und zeigt Unterschiede in der Nutzung von Groß- und Kleinschreibung sowie Abkürzungen auf.

Button Concordance Tool analysiert speziell Schaltflächen auf deren einheitliche Benennung, Größe, Position, Schriftsatz, und Farbgebung.

Button Layout Table analysiert Schaltflächen auf die konsistente Verwendung bestimmter Gruppierungen von speziellen Schaltflächen, wie z.B. [*OK, Cancel, Close, Exit, Quit, Help*], [*Start, Stop, Halt, Pause, Cancel, Close, Done, End, Exit Quit*], [*Add, Remove, Delete, Copy, Clear, Cancel, Close, Exit*] oder [*Help, Close, Cancel, Exit*].

Interface Speller überprüft die Rechtschreibung in den textuellen Inhalten der Dialoge.

Terminology Basket untersucht, ob bei der Zuweisung von Beschriftungen zu Funktionen Abweichungen auftreten. So können z.B. für die Entfernung eines Elementes die folgenden idiosynkratischen Begriffe genutzt werden: *Remove, Delete, Clear*. Mithilfe von Synonym-Listen kann die parallele Verwendung solcher Begriffe identifiziert und eine Vereinheitlichung vorgeschlagen werden.

Mahajan [Mah96] berichtet von einer Anzahl von Benutzertests zur Evaluation der Konsistenzbewertung. Diese wurde in erster Linie in Zusammenarbeit mit Designern durchgeführt und zeigten, wie SHERLOCK bei der Verbesserung von Benutzerschnittstellen eingesetzt werden kann. Analysen der Verbesserung bei der Benutzbarkeit (Performanz), bzw. beim Transfer von Wissen wurde nicht berücksichtigt. Die Nutzung von Expertenurteilen zur Validierung der Konsistenzverbesserung und somit zur Validierung der Metriken ist jedoch problematisch. Es fällt auf, dass SHERLOCK einige der Metriken aus dem vorhergehenden Abschnitt verwendet und diese auf Ähnlichkeit vergleicht.

Um nicht auf eine rein pragmatische Definition der Konsistenz zurück zu fallen, wie dies z. B. bei der Intelligenz der Fall ist (*Intelligenz ist das, was ein Intelligenztest misst*), muss ein Außenkriterium der Konsistenz gefunden werden, gegen welches ein Werkzeug zur Messung der Konsistenz verglichen werden kann. Die Validität der Konsistenzmessung kann nur im Vergleich mit messbaren Phänomenen konsistenter Darstellung evaluiert werden. In Abschnitt 7.3 werde ich auf diese Problematik zurückkommen.

SHERLOCK zeigt viele interessante Möglichkeiten auf, wie Konsistenz in einem erfasst werden kann. Jedoch ist hierbei auch deutlich geworden, dass SHERLOCK stark auf das Vergleichskriterium *Ähnlichkeit* abzielt. Elemente müssen bezüglich eines Kriteriums ähnlich sein und sind damit konsistent. Diese Sichtweise ist möglich, kann jedoch die Komplexität der plattformübergreifenden Konsistenz nicht ausreichen abbilden. Es ist unbestreitbar, dass Oberflächen auf einem Mobilgerät mit beschränkten graphischen Möglichkeiten nicht in derselben Weise dargestellt werden kann wie auf einem Desktop Computer, dennoch muss ein System die Erfassung der Konsistenz zwischen verschiedenen Plattformen ermöglichen, um den hier gestellten Anforderungen gerecht zu werden.

4.5 Zusammenfassung und Bewertung

Betrachtet man die hier vorgestellten Ansätze, bezüglich ihrer Übereinstimmung mit den in Abschnitt 2 aufgestellten Anforderungen an die Benutzbarkeit so stellt findet man sehr leicht die Aufspaltung, welche Trætteberg [Tra04] zu seiner Kritik veranlasste.

Während die toolkitbasierten Systeme nicht in dem Maße in der Lage sind sich an die Gegebenheiten verschiedener Plattformen anzupassen, bzw. die Unterstützung hierbei für den Entwickler gering ist, stellen modellbasierte Ansätze wie Beschreibungssprachen und UIMS insbesondere den Entwicklern eine große Hürde. Ebenfalls ist die Flexibilität von UIMS bezüglich der technischen Plattform

und deren Ressourcen bei UIMS und Beschreibungssprachen gering. Beschreibungssprachen haben aufgrund ihres deklarativen Charakters allerdings bereits den meisten Anklang in den Standardisierungsgremien gefunden (XForms, V2, UIML).

Die *Wissenskontinuität* ist bei toolkitbasierten Ansätzen, ebenso wie die *Adaptivität*, die *Multimodalität*, mit den Ausnahmen von [SS95, CGB00] gering. Diese kann meist nur durch die manuelle Anpassung durch den Entwickler erfolgen und ist damit fehleranfällig und leicht inkonsistent. Toolkits orientieren sich meist an den Plattformspezifika und die Abbildung lässt sich in einem gewöhnlichen Abbildungsprozess häufig schwer realisieren. Insbesondere UIMS aber auch Beschreibungssprachen erlauben aufgrund der komplexen Modellierung und mehrstufigen Anpassung meist eine gute Erfüllung der *Benutzbarkeitsanforderungen*. Insbesondere die Frage der Aufgabenkonsistenz kann in UIMS aufgrund der regelmäßigen Abbildung gut unterstützt werden [MPS04].

Angesichts heute üblicher Methoden, den schnellen Produktzyklen und der Verwendung agiler Entwicklungsmethoden [GMR04] sind die *Anforderungen an den Entwicklungsprozess* ein wichtiges Kriterium für den Erfolg und den Nutzen plattformübergreifender Entwicklungsprozesse. Die Erlernbarkeit und Verwendung von Standards ermöglicht dem Entwickler in erster Linie die Wiederverwendung von angeeignetem Wissen. Dies ist insbesondere bei Toolkits möglich, das die Objektmodelle über verschiedene Toolkits sehr einheitlich sind. Im Falle der .Net Plattform ist eine identische, wenn auch verringerte API für verschiedene Plattformen verfügbar.

Die *Verfügbarkeit von graphisch-interaktiven Werkzeugen* ist im Bereich der heute üblichen Toolkitbasierten Entwicklung eine wesentliche Voraussetzung, nicht so bei Beschreibungssprachen, die wie im Fall von XForms und UIML zwar visuelle Designer bieten, diese aber in der Regel akademische eingeschränkte Versionen sind, welche nur eingeschränkt für den professionellen Einsatz geeignet sind, eine Tatsache, welche sich entsprechend auf die *Benutzbarkeit der Werkzeuge* auswirkt.

Toolkitbasierte Methoden haben ebenfalls den Vorteil, dass sie sich in existierende existierende Entwicklungsmethoden besser einbinden lassen, da diese meist bereits toolkitbasiert sind. Modellbasierte Methoden erfordern einen hohen Grad an Abstraktion und die Transformationsschritte von den Modellen hin zur Darstellung komplizieren das Vorgehen und sind ungewohnt. Gleichwohl kommt im Rahmen der *Rational Unified Process* [Kru04a] und der *Unified Modelling Language* [BJR98] auch immer stärker modellbasierte Methoden zum Einsatz, so dass dieser zumindest möglich sein sollte. Die *transparente Behandlung der bekannten Problematiken* wie Mapping, Reversibilität, Optimierung und Konsistenz werden von Toolkitbasierten Ansätzen nicht geboten, hierfür eignen sich in erster Linie UIMS, die hierfür Methoden in den Reifikationsprozess implementieren.

Die *Erweiterbarkeit* und *Anpassbarkeit* toolkitbasierter Methoden kann von modellbasierten Methoden kaum erreicht werden, da hier eine Erweiterung auf einer Ebene (z. B. um ein abstraktes Interaktionselement) entsprechende Anpassung der gesamten Verarbeitungspipeline mit sich zieht. Der langwierige Erzeugungs- und Reifikationsprozess ist es, der eine spontane *Testung* anhand eines Prototypen ein schweres Unterfangen macht. Die *Sichtbarkeit* einer Verbesserung setzt sich von der Sichtbarkeit der Vereinheitlichung und Eingeschränktheit ab, indem hier eindeutig ein Attraktivitätsgewinn gefordert ist. Dieser ist aufgrund der notwendigen Vereinheitlichung bei der plattformübergreifenden Entwicklung in keinem der Ansätze wirklich möglich. Toolkitbasierte Lösungen durch die Anpassung noch am wahrscheinlichsten solche Ergebnisse erreichen.

Die *technische Anforderungen der Plattformunabhängigkeit* wird von jedem der Systeme inhärent erfüllt. Jedoch betrachtet man die notwendigen Erzeugungsprozesse, insbesondere bei serverbasierten Ansätzen, das wird klar, dass zumindest Anforderungen an die Konnektivität und die Verfügbarkeit von Infrastrukturen gestellt werden, diese schränken in der Regel auch die *Skalierbarkeit* ein. Die Dynamische Konfiguration von toolkitbasierten und UIMS Systemen ist recht gut, da hier auf viele der Parameter direkt in der Laufzeitumgebung zugegriffen werden kann, die Beschreibungssprachen stellen meist statische Ressourcen dar und können nicht dynamisch angepasst werden. Beschreibungssprachen haben den größten Anteil an Standards, Toolkits verfügen jedoch meist auch über einheitliche Objektmodelle.

Aus dieser zusammenfassenden Betrachtung kann für ein zu realisierendes System zur Unterstützung der Entwicklung plattformübergreifender Benutzerschnittstellen abgeleitet werden, dass dieses zum einen in Adaptierbarkeit und Benutzbarkeit das Niveau modellbasierter Ansätze erreichen sollte, zum anderen aber auch die Flexibilität und Einfachheit toolkitbasierter Methoden zur Verfügung

	Anforderung	Toolkits	Beschreibungssprachen	UIMS
Benutzbarkeit	Wissenskontinuität	-	+	+
	Kontextadaptivität	-	+	+
	Multimodalität	-	+	+
	Benutzbarkeit	-	-	+
Entwicklung	Erlernbarkeit	+	-	-
	Werkzeuge	+	-	+
	Einbindung	+	-	-
	Transparenz	-	+	+
	Erweiterbarkeit	+	-	-
	Testbarkeit	+	-	-
	Sichtbarkeit	-	-	+
Technik	PF-Unabhängigkeit	-	+	+
	Systemressourcen	+	-	-
	Dynamisch	+	-	+
	Standards	-	+	-

Tabelle 4.4: Bewertung der wichtigsten Ansätze bezüglich der Anforderungen an plattformübergreifende Benutzerschnittstellen und deren Entwicklung.

stellen sollte. Existierende Ansätze ordnen sich stets einer der beiden Gruppen zu und erfüllen diese Anforderungen nicht.

Die Entwicklung sollte durch entsprechende Hilfen und automatische Methoden geleitet und bezüglich der plattformübergreifenden Benutzbarkeit unterstützt werden. Neben den Anforderungen der Modalitätenanpassung und der Anpassung an Kontextparameter wie Gerät, Nutzer, Umgebung muss hier also die Verbesserung der Benutzbarkeit im Sinne der Wissenskontinuität und anderen ergonomischen Aspekten im Vordergrund stehen. Dies soll durch Methoden der automatischen Evaluierung der Oberfläche und deren Integration in den Entwicklungsprozess ermöglicht werden.

Der klare Vorteil toolkitbasierter Ansätze gegenüber modellbasierten Ansätzen bezüglich der Akzeptanz und Anwendbarkeit in professionellen Entwicklungsprozessen spricht dafür zumindest für den Benutzer toolkitbasierte Methoden anzubieten. Um die Adaptionleistung zu erreichen muss jedoch eine Abstraktionsschicht eingezogen werden, die sich dem Entwickler jedoch transparent verhält. Die Kompatibilität zu existierenden Werkzeugen sollte berücksichtigt werden. Die Berücksichtigung graphischer Methoden sollte gegeben sein.

Ein weiterer Vorteil toolkitbasierter Methoden soll durch eine hybride Architektur (Kapselung einer Abstraktionsschicht und Toolkit API) mit den Vorteilen plattformunabhängiger UIMS verbunden werden, der geringe Ressourcenverbrauch kompakter Toolkits. Die Verwendung von Standards führt zur möglichen Integration in existierende Werkzeuge und garantiert die Stabilität der technischen Grundlagen.

Das System, welches hier entwickelt werden soll, wird basierend auf dieser Bedarfsanalyse in erster Linie die Entwicklung eines Entwicklungsmodells adressieren, welches mit toolkit- und modellbasierten Methoden kompatibel ist. Weiterhin wird die Möglichkeit der Evaluation der Benutzbarkeit graphischer Oberflächen und die Assistenzmöglichkeiten bei deren Entwicklung betrachtet.

5 Ein Konzept plattformübergreifender Entwicklung

Das Referenzmodell des Entwicklungsprozesses hat zum Ziel, die in Abschnitt 2 gewonnenen Anforderungen zusammen mit den Grundlagen, welche in Abschnitt 3 vorgestellt wurden, zu verbinden und darauf aufbauend einen integrierten, flexiblen und benutzerzentrierten Entwicklungsprozess darzustellen. Die spezifischen Probleme der existierenden Ansätze, welche in Abschnitt 4 ausführlich beschrieben wurden, sollen hierbei diskutiert und alternative Lösungsansätze vorgeschlagen werden.

Der hier vorgestellte Ansatz basiert auf einigen *Grundannahmen*, welche hier zunächst nochmal einmal aufgegriffen werden sollen.

Anwendungs-, Geräte- und plattformübergreifende Konsistenz: Die Konsistenz eines Systems, zumindest soweit diese sich auf zielgerichtete Funktionalitäten bezieht [Gru89], wird als eine wichtige Voraussetzung für die Benutzbarkeit eines Systems betrachtet (s. hierzu auch Abschnitt 3.3).

“Consistency is assumed to enhance the user’s possibility for transfer of skill from one system to another. By doing so, consistency leads to ease of learning and ease of use.”
[Nie89, S. 4]

Aus diesem Grunde soll die Konsistenz ein zentrales Gestaltungsmerkmal bei der Entwicklung plattformübergreifender Benutzerschnittstellen darstellen. Der Zusammenhang zwischen der effektiven Nutzbarkeit und Wiederverwendbarkeit des mentalen Modells (s. Abschnitt 3.2) und der Konsistenz, als Beziehung zwischen den u. U. in Konflikt stehenden Modellen zweier Anwendungen [Rei90], wird durch einen möglichen Wechsel zwischen Plattformen noch zusätzlich kompliziert. Hier ist zum einen die Konsistenz innerhalb einer Anwendung selbst (*intra-application*), weiterhin die Konsistenz derselben Anwendung auf verschiedenen Plattformen *inter-device*, sowie nicht zuletzt die Konsistenz innerhalb der Plattform, d. h. die Einhaltung von Plattform-Standards (*intra-device*), zu beachten.

Die automatische Unterstützung bei der Überprüfung der Konsistenz einer Anwendung wurde z. B. von Mahajan und Shneiderman [MS97] mittels des SHERLOCK Konsistenzüberprüfungswerkzeuges realisiert und erfolgreich getestet. Das Referenzmodell berücksichtigt die Einhaltung der Konsistenz auf mehreren Ebenen, indem zum einen die Darstellung aus derselben abstrakten Spezifikation gewonnen wird, zum anderen eine Darstellung zunächst auf Plattformstandards aufbaut und zuletzt eine Evaluation der verschiedenen Darstellungen bezüglich ihrer Übertragbarkeit, bzw. Konsistenz erfolgt.

Interdisziplinäre benutzerzentrierte Entwicklung: Eine der wichtigsten Anforderungen an die Entwicklung von Benutzerschnittstellen ist die Berücksichtigung und Integration der Benutzbarkeit als dem zentralen Qualitätsmerkmal interaktiver Systeme. Um die Benutzbarkeit des Produktes zu gewährleisten, muss diese während des gesamten Entwicklungsprozesses einbezogen werden.

Im Gegensatz zu primär technisch motivierten Vorgehensmodellen wie dem CAMELEON Referenzmodell [TCC04], dem UIML-Modell von Abrams *et al.* [FAPQA04], dem *Representation Framework* von Puerta und Eisenstein [PE04], oder kognitionstheoretisch motivierten Modellen wie dem MASTERMIND-Modell von Brown *et al.* [BDRS97], wird hierzu ein benutzerzentriertes Vorgehensmodell benötigt, welches sich an existierenden Entwicklungsprozessen orientiert. Im Vordergrund muss der Entwicklungsprozess, und das Entwicklerteam als Benutzer stehen. Von zentraler Bedeutung ist weiterhin die Einbindung von Evaluations- und Kommunikationsprozessen (s. auch Abschnitt 2.2) mit dem Ziel der Verbesserung der Benutzbarkeit.

Aus diesem Grunde orientiert sich das hier vorgestellte Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen an einer etablierten benutzerzentrierten Entwicklungsmethodik, dem *Usability Engineering Lifecycle* von Mayhew [May99]. Während Mayhews *Lifecycle* in erster Linie als Leitfaden und methodische Vorgehensweise in interdisziplinären Entwicklungsteams gedacht ist, werden in dem vorliegenden Modell die Phasen der Entwicklung und Darstellung einer

plattformunabhängigen Benutzerschnittstelle klar spezifiziert, konkretisiert und zu einem Referenzmodell des Entwicklungsprozesses ausgearbeitet.

Vom Entwicklungswerkzeug zum Entwicklungsassistent: Eine benutzerzentrierte Sichtweise bedeutet in einem Entwicklungssystem insbesondere die Berücksichtigung des Entwicklers als dessen Benutzer. Wie bereits in den Anforderungen in Abschnitt 2.4.2 ausgeführt wurde, soll die Unterstützung über die Ebene des reinen Werkzeugs hinaus gehen und dem Entwickler interaktive Assistenz bieten. Die Komplexität des Entwicklungsprozesses soll durch Assistenz auf verschiedenen Ebenen der Mensch-Maschine-Interaktion [ND86] erfolgen.

Motivation, Aktivierung und Zielformulierung Der Entwickler wird zu *gutem* Design ermutigt und über strukturierte Methoden geleitet.

Wahrnehmung Problematische Designentscheidungen und Inkonsistenzen, die in gängigen Werkzeugen meist nicht dargestellt werden, werden visualisiert und damit wahrnehmbar gemacht.

Informationsintegration und Situationsbewusstsein Algorithmen helfen, beobachtbare Zustände zu integrieren und zu bewerten.

Entscheidung und Handlungsplanung Methoden zur Gestaltung werden an die Evaluierungsmethoden gekoppelt, so dass die Übereinstimmung der Handlung mit den Handlungszielen gewährleistet wird

Ausführung Automatisierung verschiedener Weise (Delegation, Übernahme und Informativ) ebenso wie motorische Führung sind ein Gestaltungsziel der Unterstützungsmethoden.

Rückmeldung und Bewertung Durch die Fokussierung auf direkt-manipulative Methoden ist eine direkte Kopplung an Handlung und Rückmeldung und damit eine direkte Bewertung möglich.

Die hier formulierten Grundlagen werden in den Konzepten der benutzerzentrierten Entwicklung und Entwicklungsunterstützung des Designprozesses plattformübergreifender Benutzerschnittstellen berücksichtigt werden.

5.1 Benutzerzentrierte Entwicklung

Mayhew [May99] ist es gelungen mit dem *Usability Engineering Lifecycle* die Anforderungen moderner Methoden des *Usability Engineering* in einen objektorientierten Softwareentwicklungsprozess (*Object-Oriented Software Engineering*, OOSE) nach Jacobson *et al.* [Jac92] einzubinden.

Der OOSE teilt sich auf in drei Phasen: einer *Analysephase*, einer *Konstruktionsphase* und der *Phase der Testung*. In der Analysephase werden die Anwendungsfälle (*Use Cases*) und die Akteure identifiziert und darauf aufbauend ein Anforderungsmodell entwickelt. Dieses Anforderungsmodell wird dann aus Systemsicht interpretiert und in systemkonforme Komponenten im sogenannten Analysemodell beschrieben. In der Konstruktionsphase wird dann das Design- und das Implementierungsmodell entwickelt. Zuletzt werden in der Phase der Testung das Design und die Implementierung überprüft und wenn nötig in einem weiteren Zyklus überarbeitet.

Ein wichtiger Aspekt ist hierbei das Vorgehen vom Abstrakten zum Konkreten, d. h. der Entwicklung *top-down*, sowie einem *iterativen* Vorgehen, in dem sich Phasen der Konstruktion und der Testung regelmäßig abwechseln, um so eine möglichst enge Kopplung des Entwicklungsprozesses an dessen Verifizierung und Validierung zu gewährleisten und somit eine hohe Integration evaluativer Methoden zu erreichen.

Ziel des *Usability Engineering* ist es, Methoden zu entwickeln und bereitzustellen, welche es ermöglichen, *Usability* Aspekte an zentraler Stelle in den Softwareentwicklungsprozess einzubinden. Denn, wie Karat und Dayton [KD95] feststellen, existiert ein zentrales Problem heutiger Methoden in der Tatsache, dass „*in most cases of the design and development of commercial software, usability is not dealt with at the same level as other aspects of software engineering*“. Mayhew stellt fest, dass eine Organisation zur Entwicklung benutzbarer Produkte zumindest zwei Dinge benötigt [May99]:

- Wissen über bekannte Gestaltungsregeln und Leitfäden und deren Anwendung.

- Wissen über strukturierte Methoden zur Erreichung von Benutzbarkeit in Produkten und deren Umsetzung.

Dies bedeutet zum einen, dass eine Organisation Mitarbeiter benötigt, welche Erfahrung in der Gestaltung benutzbarer Produkte besitzen, welchen die wichtigsten Gestaltungsregeln bekannt sind und welche in der Lage sind, diese auch umzusetzen. Zum anderen muss die Organisation Prozesse implementiert haben, welche die kooperative Entwicklung unterstützen und koordinieren. Von zentraler Bedeutung ist hierbei, dass der Prozess der Entwicklung von Benutzerschnittstellen in der Regel nicht von einzelnen Personen getragen werden kann, sondern interdisziplinär und kooperativ erfolgt.

Seit den ersten Ansätzen des *Usability Engineering* in den frühen siebziger Jahren, wurde eine Vielzahl von Methoden und Vorgehensmodelle entwickelt: unter anderem der *Usability Engineering Lifecycle* von Nielsen [Nie92, Nie93] oder die *Three Pillars of Design* von Shneiderman [Shn98]. Gould und Lewis [GL83] beschreiben in ihrem frühen Ansatz die folgenden drei globalen Strategien des Usability Engineering:

- Frühe Berücksichtigung des Benutzers und der Aufgaben.
- Empirische Überprüfung.
- Iteratives Design.

Mayhew teilt den benutzerzentrierten Entwicklungsprozess in drei Phasen. Eine dieser Phasen, die des Designs, der Testung und Entwicklung ist wiederum in drei Phasen aufgeteilt. Diese Phasen werden in der Folge detailliert beschrieben (s. Abbildung 5.1).

Anforderungsanalyse In der Phase der Anforderungsanalyse werden die aufgabenrelevanten Konzepte erfasst und in Form von Anforderungen an das System formuliert. Das Ergebnis ist eine Spezifikation des Systems auf einer abstrakten konzeptuellen Ebene. Hauptgegenstand dieser Phase sind:

Benutzerprofil Beschreibung der für das System relevanten Eigenschaften des Benutzers und Variationen dieser Eigenschaften sowie deren Bandbreite.

Aufgabenanalyse Analyse der aktuellen Aufgabenbearbeitung, Dekomposition und wo möglich, Optimierung.

Ziele bzgl. der Benutzbarkeit Qualitative Ziele, die in Bezug auf die erfassten Benutzer und der Fähigkeiten formuliert werden und eine optimale Passung von Systemkomplexität und Benutzerfähigkeiten sowie Arbeitskontext ermöglichen sollen.

Technische Anforderungen Anforderungen an die Plattform bzgl. Systemleistung, möglicher Interaktionsmetaphern, etc.

Design, Testung und Entwicklung Diese Phase des Entwicklungszyklus umfasst tatsächlich drei Unterphasen, welche jeweils in iterativer Form jeweils Design, Testung und Entwicklung beinhalten. Diese Unterphasen bauen aufeinander auf und umfassen den eigentlichen technischen Entwicklungsprozess.

Konzeptuelle Phase Die konzeptuelle Phase umfasst zunächst die Analyse und Formalisierung der im ersten Schritt erfassten *Arbeitsprozesse* und deren Segmentierung in implementierbare Unterschritte. Hierauf aufbauend werden *konzeptuelle Modelle* entwickelt, welche eine erste Realisierung der Navigationspfade und Darstellung auf einer sehr allgemeinen wenig detaillierten Ebene darstellen, und welche dann in der Folge in einem ersten Schritt iterativ *evaluiert* und *verbessert* werden.

Screen Design Standards In dieser Phase wird das konzeptuelle Modell auf die nächste Ebene der Konkretisierung gehoben, indem die entwickelten Modelle nun in der, für die Zielplattform typischen Weise, d. h. den gängigen Standards oder Guidelines entsprechend, umgesetzt werden. Die Realisierung erfolgt direkt, ohne spezifische Anpassungen und hat zum Ziel eine *konsistente* und *plattformkonforme Darstellung* zu gewährleisten. Auch diese Phase wird mit Evaluationen und Verfeinerungen in iterativen Zyklen umgesetzt.

Detailliertes Design Das Design wird in dieser Phase *auf das spezifische Produkt angepasst* und für dessen Zwecke detailliert und optimiert. Aufbauend auf der plattformkonformen Vorstufe können nun detaillierte Veränderungen, so weit wie möglich im Rahmen der Plattformstandards umgesetzt und anschließend evaluiert werden.

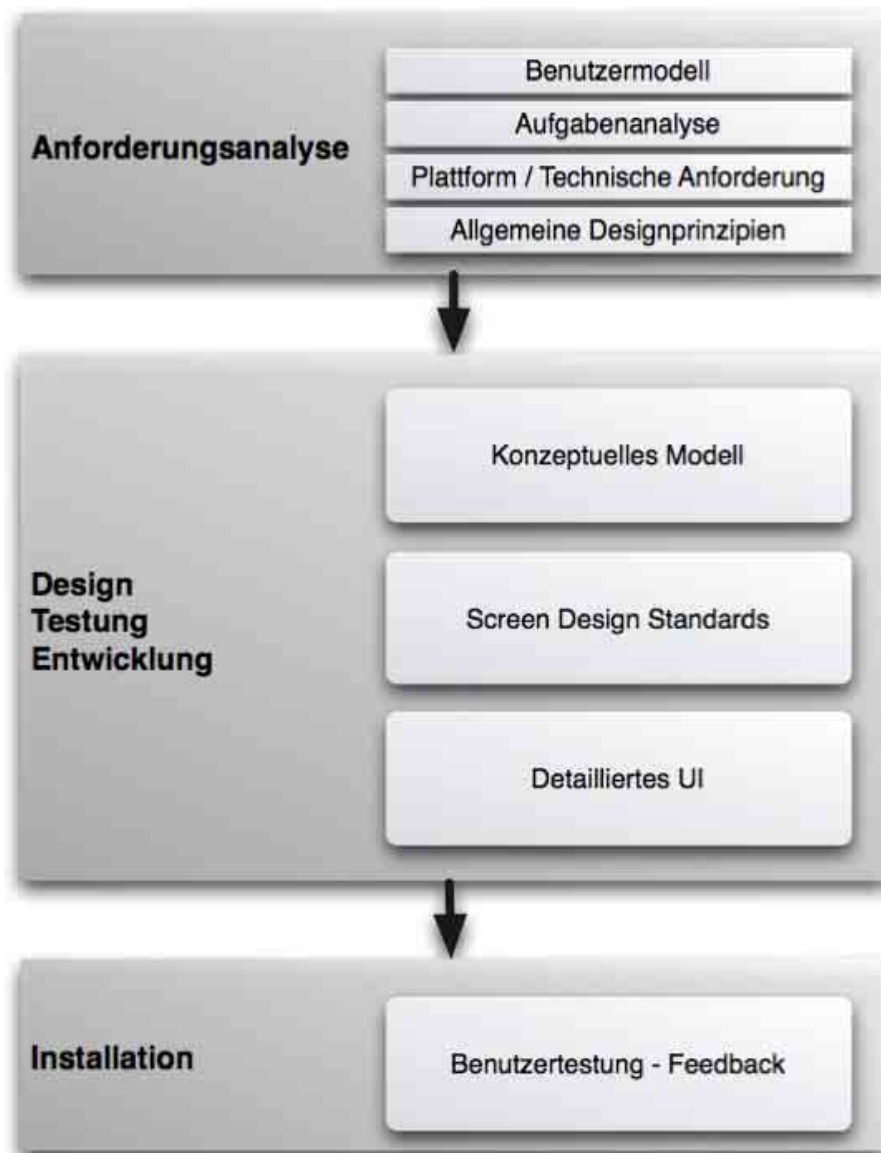


Abbildung 5.1: Der Usability Engineering Lifecycle von Mayhew [May99].

Installation Die Phase der Installation ist auf die langfristige Produktpflege ausgerichtet und beinhaltet in erster Linie die Integration von Benutzerfeedback und Erfahrungen im produktiven Einsatz in die Produktentwicklung und Überarbeitung.

Das *Usability Engineering Lifecycle* Modell zeichnet sich dadurch aus, dass es einen sehr allgemeinen Gültigkeitsbereich besitzt und den Entwicklungsprozess in interdisziplinären Teams mit dem Fokus der Benutzerzentrierung und Benutzbarkeit unterstützt. Das Vorgehen orientiert sich hierbei nicht an technischen Vorgaben, sondern an den existierenden Prozessen bei der Entwicklung von Softwareprodukten und versucht diese durch methodische Vorgaben zu leiten und zu unterstützen.

Betrachtet man die in Abschnitt 2.4 gesammelten Anforderungen an den Entwicklungsprozess, erscheint dieses Modell als eine geeignete Basis für ein spezifisches Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen. Gleichwohl werden spezifische Anforderungen und Probleme, wie sie in Abschnitt 2.4 erfasst wurden, nicht adressiert, so bietet dieses Modell keine Lösung für die parallele Entwicklung von verschiedenen Darstellungen (*Konsistenz-Problematik*), die Rückführung von konkreten Spezifikationen auf abstrakte (*Mapping, Round-Trip-ping*) und die methodische Unterstützung von adaptiven Systemen. Ein Referenzmodell der plattformübergreifenden Entwicklung muss diese Aspekte aufgreifen und integrieren.

5.2 Referenzmodell der plattformübergreifenden Entwicklung

Aufbauend auf dem *Usability Engineering Lifecycle* soll nun ein benutzerzentriertes Entwicklungskonzept für plattformübergreifende Benutzerschnittstellen definiert und beschrieben werden. Dieses Modell erweitert und verfeinert das *Lifecycle* Modell um Aspekte, welche bei der Entwicklung plattformübergreifender Benutzerschnittstellen von Bedeutung sind. Analog zu dem *Lifecycle* Modell findet die Entwicklung der Benutzerschnittstelle *top-down*, d. h. von der abstrakten Spezifikation oder Modellen, hin zur konkreten Darstellung statt. Da die plattformübergreifende Entwicklung jedoch nicht auf eine Darstellung, sondern beliebige Darstellungen in Abhängigkeit von der verwendeten Plattform, hinarbeitet, muss diese mehrfache Abbildung im Entwicklungsprozess berücksichtigt und unterstützt werden.

Folgende Eigenschaften des Referenzmodells sind hierbei von besonderer Bedeutung:

- Das Referenzmodell definiert *geordnete Entwicklungspfade* (*ordered-path development*) anstatt der ungeordneten (*multi-path development*), teils parallelisierten Entwicklungspfade in anderen Modellen. Dies bedeutet, dass ein Entwicklungspfad — der primäre Entwicklungspfad — die Umsetzung dominiert, d. h. die Darstellung wird zunächst für eine Plattform ausgeführt, diese dient als Referenz. Darstellungen für andere Plattformen leiten sich lediglich von dieser Darstellung ab, diese stellen dann den sekundären Entwicklungspfad dar. Dieses Vorgehen leitet sich aus dem tatsächlichen Vorgehen von Designer ab welcher die Gestaltung in der Regel anhand einer Referenzvorlage realisiert und nicht alle möglichen Abbildungen parallel entwickelt.
- Das *Transformationsparadigma* fasst die oben beschriebene Vorgehensweise der geordneten Entwicklung in ein Systemprinzip und dient als Grundlage für das Konzept der Konsistenz zwischen den Plattformen, anhand dessen die Qualität einer Ableitung systemgestützt evaluiert und u. U. automatisiert optimiert werden kann. Betrachtet man die Darstellung auf einer sekundären Plattform als Ableitung, bzw. Transformation, von der primären, so ist das Maß der — wie auch immer formalisierten — Ähnlichkeit zwischen den Darstellungen ein Qualitätsmaß für die Umsetzung.
- Die Konkretisierung, und die dabei erfolgende Anpassung der Darstellung an die Zielplattform und andere Kontextparameter erfolgt in dem vorgelegten Modell sequenziell. Analog zu dem *Lifecycle* Modell wird zunächst auf fest implementiert Standardprozesse zurückgegriffen bevor dann in einem zweiten Schritt individuelle und u. U. zeitnahe Anpassungen erfolgen. Diese *Sequenzielle Anpassung* dient in erster Linie der Lösung des *Mapping*, bzw. *Round-Tripping* Problems, indem eindeutige Abbildungen möglich werden und somit die Umkehrbarkeit der Konkretisierung ermöglicht wird. Wie in den Anforderungen erwähnt, ist dies vor allem dann notwendig, wenn eine Bearbeitung auf einer konkreteren Darstellungsebene möglich sein soll.

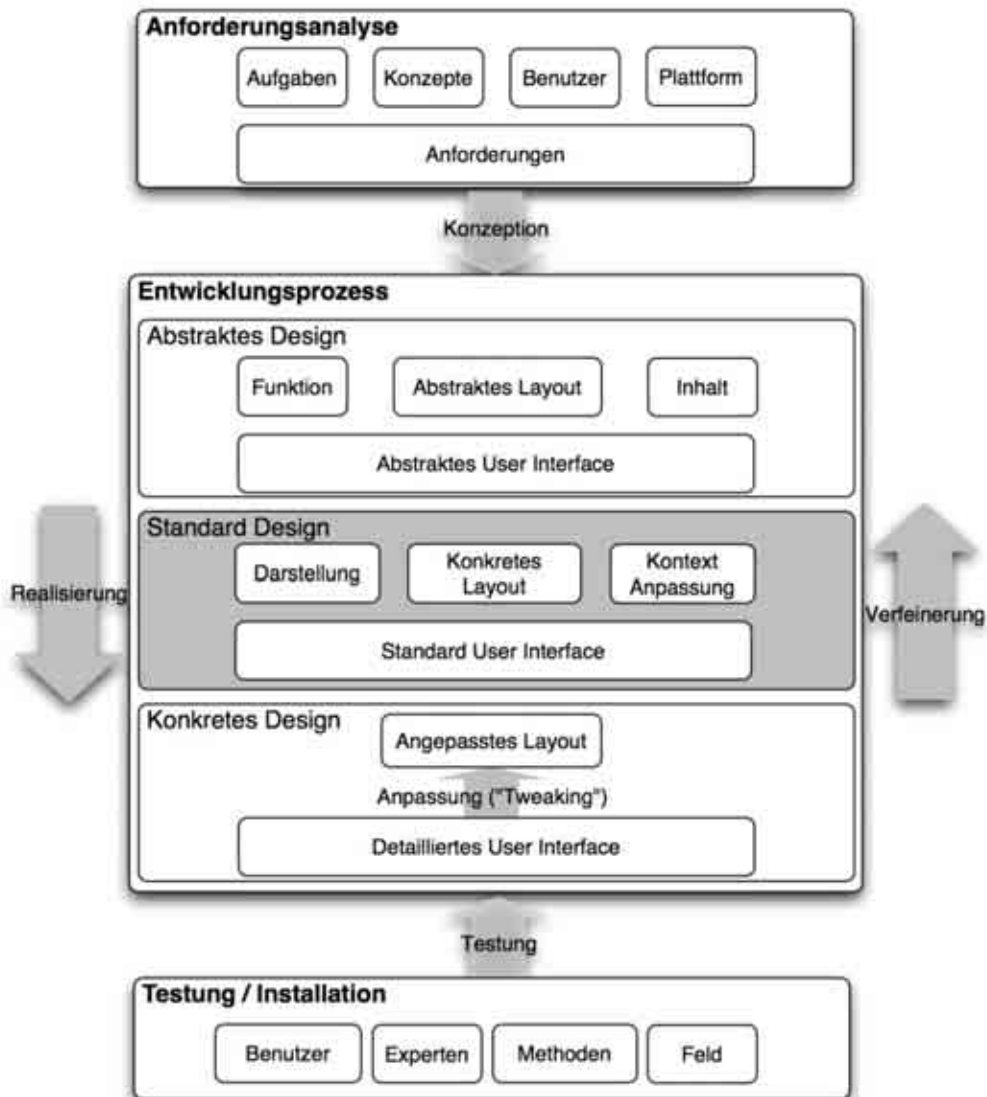


Abbildung 5.2: Das Referenzmodell der Entwicklung plattformübergreifender Benutzerschnittstellen

Abbildung 5.2 zeigt eine Übersicht des Referenzmodells. Dieses Modell bildet die drei Stufen des *Usability Engineering Lifecycles* auf den Entwicklungsprozess ab, wobei der Entwicklungsprozess in die Anforderungsanalyse und die Testung/Installation eingebettet ist. Prinzipiell wird in diesem Modell der Prozess der Realisierung oder Konkretisierung, d. h. hin zur konkreten Darstellung von dem Prozess der Verfeinerung, d. h. der Anpassung der Konkreten Darstellung durch Korrektur der abstrakteren Repräsentationen, unterschieden. In Abbildung 5.2 nicht enthalten ist die Differenzierung in die verschiedenen Entwicklungspfade, diese wird der Übersichtlichkeit zuliebe an späterer Stelle (s. Abschnitt 5.2.2) ausführlich behandelt werden.

Im folgenden Abschnitt werden nun die einzelnen Phasen des Entwicklungsmodell genauer beschrieben.

Anforderungsanalyse Die Anforderungsanalyse umfasst den Prozess der Sammlung und Spezifikation der erwünschten Systemeigenschaften, der Kontextparameter auf die das System anzupassen sein soll und die eigentliche Funktionalität. Die Anforderungen gehen dann in Form von formalen Modellen oder einfachen Anforderungskatalogen in den eigentlichen Entwicklungsprozess ein und dienen diesem als Ausgangsinformation.

Entwicklungsprozess Der Entwicklungsprozess ist der eigentliche Kernprozess und umfasst drei Unterprozesse. In diesem Prozess wird die Konkretisierung der abstrakten Spezifikation hin zu einer oder mehreren plattformspezifischen Darstellungen umgesetzt, sowie die Verfeinerung und Anpassung der Darstellungsprozesse durch den Designer und Usability Experten eingebracht.

Abstraktes Design Das Abstrakte Design umfasst die plattformunabhängige Spezifikation der Benutzerschnittstelle, das *Abstrakte User Interface*. Diese wurde aus den Anforderungen entweder durch den Entwickler manuell oder auf Basis automatischer Prozesse im Sinne der modellbasierten User Interface Entwicklung (s. Abschnitt 4) generiert. Die Abstrakte Spezifikation der Oberfläche ist eine ausreichende Basis für die Erzeugung der Darstellungen in allen Plattformen.

Standard Design Das Standard Design umfasst die Prozesse, welche aus der Abstrakten Darstellung eine erste, allgemeine und standardmäßige Darstellung, das *Standard User Interface* erzeugen. D. h. feste Umsetzungsprozesse müssen implementiert sein, welche die abstrakte Spezifikation für eine Plattform umsetzen können. Hierbei werden in der Regel anwendungs- und kontextunabhängige Umsetzungen vorgenommen, d. h. einer abstrakten Beschreibung wird eine Systemstandard gemäße konkrete Darstellung zugeordnet. Diese Darstellung ist dann natürlich weder anwendungs- noch kontextbezogen optimiert noch bietet dieser Prozess die Möglichkeit einer Optimierung nach Usability- oder Designmaßstäben, er bietet dafür aber eine plattformintern konsistente und reversible Abbildung und dient als erster Anpassungsschritt.

Konkretes Design Das konkrete Design repräsentiert die endgültige Darstellung, so wie sie entweder in einem graphischen Editor, einer Entwicklungsumgebung oder dann zur Laufzeit auf dem Endgerät dargestellt wird. Diese Darstellung beinhaltet neben den, im ersten Schritt durchgeführten standardmäßigen Konkretisierungen, auch die Anpassungen, welche vom Designer oder Usability Experten im Rahmen des Design- und Testungsprozesses durchgeführt wurden und damit eine anwendungsbezogen optimierte Darstellung. Weiterhin fließen bei der Realisierung dieser Ebene ebenfalls kontextspezifische Anpassungen ein, welche in der Regel zur Laufzeit über Sensoren oder Benutzerdaten erfasst und angewandt werden.

Testung / Installation Existiert nun mit dem konkreten Design auch das *Detaillierte User Interface*, kann eine konkrete Darstellung erzeugt werden, d. h. das Endprodukt ist erreicht. Die Darstellung kann nun entweder von einem Designer, dem Software Entwickler oder einem Usability Experten untersucht, angepasst und verfeinert werden, wobei die Anpassungen dann entweder direkt in das *Abstrakte User Interface* aufgenommen werden (*Primärer Entwicklungspfad*) oder in die Anpassungsprozesse im Übergang von *Standard* zu *Detailliertem User Interface* einfließen (*Primärer* und *Sekundärer Entwicklungspfad*), oder an den Benutzer ausgeliefert werden und durch das Aufgreifen von Benutzerfeedback verbessert werden.

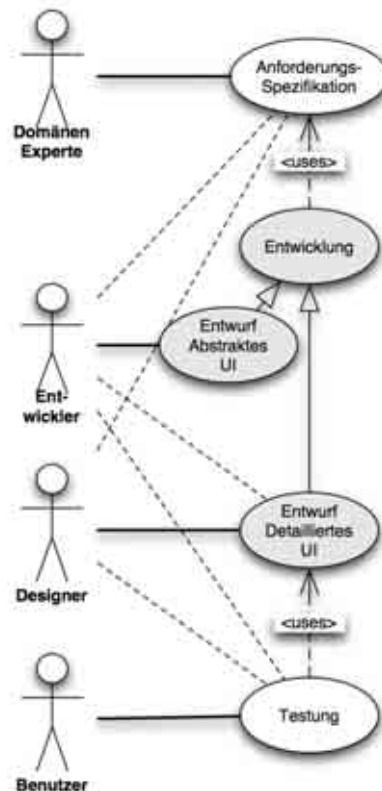


Abbildung 5.3: Bearbeitungsrollen und damit verbundene Arbeitsbereiche

Das Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen erweitert das *Lifecycle* Modell Mayhews um die Möglichkeit, mehrere parallele plattform- und kontextsensitive Entwicklungspfade zu verwalten und zugleich eine Bearbeitung der Oberflächen sowohl auf einem abstrakten konzeptionellen als auch auf einem konkreten darstellungsnahen Niveau zu bearbeiten und somit den Anforderungen der verschiedenen Bearbeiterrollen gerecht zu werden. Tatsächlich trennt das Modell, wie Abbildung 5.3 zeigt, die Bearbeitungsbereiche der verschiedenen Bearbeiterrollen auf klare Weise und orientiert sich hierbei an gängigen Vorgehensweisen im Softwareentwicklungsprozess.

In den folgenden Abschnitten wird nochmals auf die einzelnen Problematiken im Entwicklungsprozess plattformübergreifender Benutzerschnittstellen eingegangen und es wird genau dargestellt wie das Modell diese Problematiken behandelt und Lösungsansätze bereitstellt.

5.2.1 Sequenzielle Anpassung

Abbildung 5.4 verdeutlicht, wie die Darstellung einer Benutzerschnittstellen für verschiedene Plattformen realisiert wird. In einem ersten Schritt werden verschiedene Anwendungseigenschaften analysiert und in einen Anforderungskatalog überführt. Diese Anforderungen werden dann (s. u.) in eine abstrakte Beschreibung der Oberfläche umgesetzt, welche als Ausgangspunkt für die sequenzielle Anpassung dient. Plattformstandards werden als Übersetzungsregeln in eine erste plattformspezifische Darstellung genutzt. Diese wird dann anhand der plattformspezifischen Designanpassungen und die Kontextinformationen in die endgültige Darstellung übersetzt.

Die Anpassung erfolgt sequenziell in mehreren Schritten, welche jeweils eine klar definierte Rolle spielen. Diese sequenzielle Anpassung ermöglicht die kontrollierte Auftrennung des Entwicklungs- und Darstellungsprozesses in Phasen. Diese sind damit getrennt zu gestalten und zu optimieren. Be-

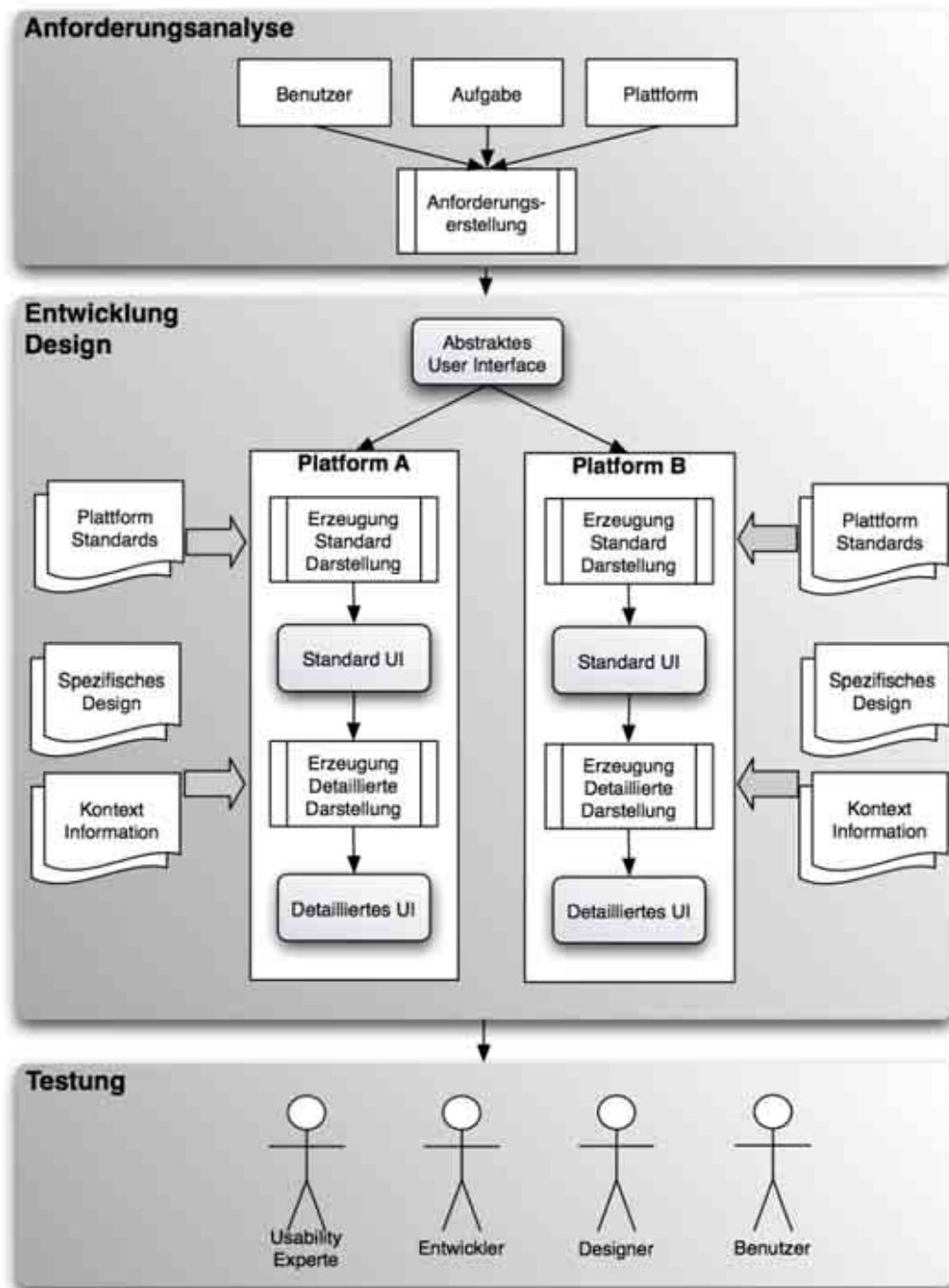


Abbildung 5.4: Der Darstellungsprozess

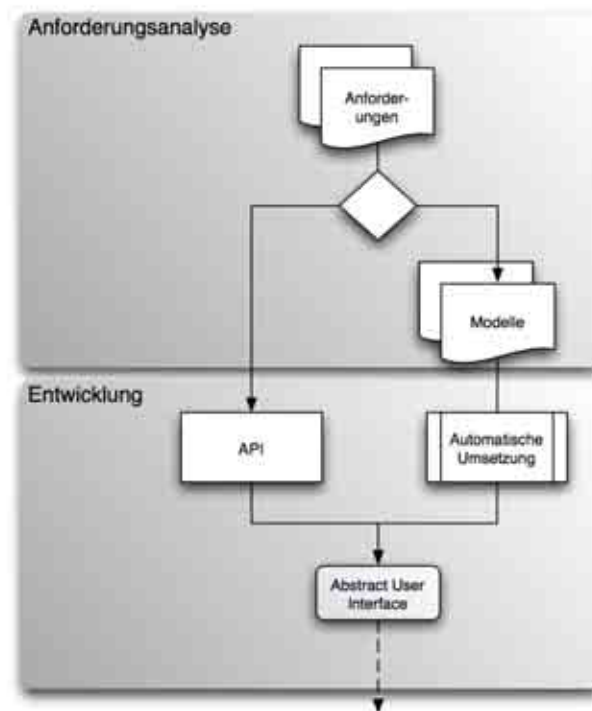


Abbildung 5.5: Ableitung des Abstrakten User Interfaces aus den Anforderungen

züglich der identifizierten Problematiken des Entwicklungsprozesses liefert dieses Vorgehen somit die Voraussetzung zur Lösung des *Round-Tripping*, bzw. *Mapping Problems*.

Die einzelnen Phasen des Darstellungsprozesses verwenden spezifische Daten um die Anpassung vorzunehmen, diese werden zum Teil speziell für ein Produkt gesammelt, wie z. B. die *Anforderungen*, die *spezifischen Designanpassungen* und *Kontextdaten*. Zum Teil sind sie auch anwendungsübergreifend einsetzbar, wie z. B. die *Plattform Standards*. Um diese Daten gezielt anpassen zu können, d. h. ein gezieltes plattformspezifisches Design oder die Entwicklung von Prozessen zur Umsetzung des Plattform-Standards zu entwickeln, müssen die einzelnen Phasen isoliert werden können.

Für den Prozess der Anforderungsentwicklung und deren Umsetzung in eine abstrakte Beschreibung der Oberfläche sieht das Modell mehrere mögliche Wege vor (s. Abbildung 5.5). Es integriert somit die existierenden konventionellen Vorgehensweisen in der Software-Entwicklung und moderne experimentelle Verfahren wie die modellbasierten Ansätze auf einer konzeptionellen Ebene und bietet somit auch die Möglichkeit, Übergänge zwischen den Methoden zu identifizieren, bzw. die schrittweise Einführung neuer Methoden in existierende Prozesse zu unterstützen.

Konventionelles Vorgehen Die Ableitung des abstrakten User Interfaces aus den Anforderungen kann, wie heute üblich, über formale oder skizzenhafte Spezifikationen durch den Entwickler erfolgen. Dieser setzt die Spezifikation im Sinne der Realisierung von Dialogen, Dialogelementen und deren Übergängen mittels einer Anwendungsprogrammierungsschnittstelle (*Application Program(ming) Interface*, API) in lauffähigen, aber darstellungs- und plattformunabhängigen Code um.

Modellbasiertes Vorgehen Im Falle der modellbasierten Entwicklung wird die Anforderungsspezifikation in Form von Modellen umgesetzt. Diese werden durch einen oder mehrere kaskadierende Prozesse in eine abstrakte Spezifikation des User Interfaces umgesetzt. Thevenin und Coutaz [TCC04, LVM⁺04] bezeichnen diese Ebene als die des *Abstract UI (AUI)*. Vorteil dieser Vorgehensweise ist, dass die konsistente Dialogstruktur, welche aus der modellbasierten Erzeugung resultiert [MPS04] dann in eine konsistente Darstellung mittels des Entwicklungsmodells überführt werden kann.

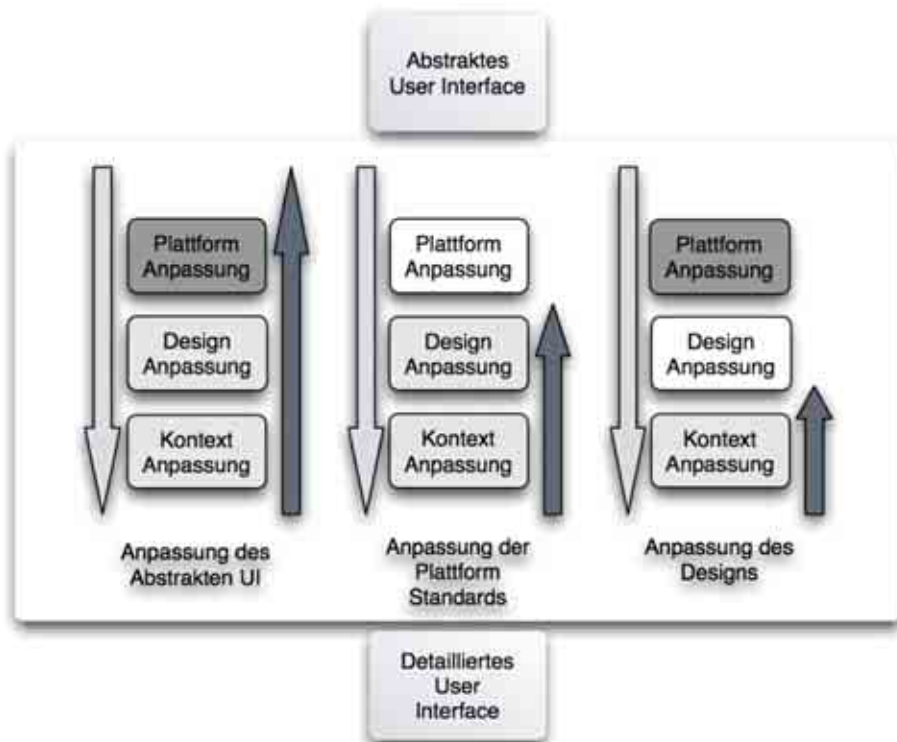


Abbildung 5.6: Die sequenzielle Anpassung des Darstellungsprozesses.

Die genauere Definition der Prozesse der *Anforderungsspezifikation* stehen nicht im Fokus dieses Modells. Hierfür sein auf das *Lifecycle* Modell von Mayhew [May99], die Methoden des *Contextual Design* von Beyer und Holtzblatt [BH98] und Methoden der Benutzer- und Aufgabenanalyse [HR98] verwiesen. Diese Prozesse gestalten sich in Bezug auf die Darstellungsprozesse relativ unabhängig und sollen in der Folge nicht detaillierter beschrieben werden. Vielmehr soll in der Folge betrachtet werden, wie aus der abstrakten Beschreibung in einem sequenziellen Anpassungsprozess eine zunehmend konkrete, plattformspezifische Darstellung entwickelt werden kann, und wie die einzelnen Anpassungsprozesse aufgrund dieses sequenziellen Aufbaus einzeln gestaltet und optimiert werden können.

Abbildung 5.6 gibt eine schematische Darstellung des sequenziellen Anpassungsprozesses. Als Basis wird das Abstrakte User Interface verwendet, dieses steht als Spezifikation am Anfang der Darstellungskette. Das Darstellungsmodell koppelt, in Entsprechung des *Lifecycle* Modells, den Prozess der Spezifikation der Anforderung vom eigentlichen Entwicklungsprozess ab. Dies ist notwendig, da *zum einen* der Grad der Formalisierung dieses Prozesses variiert. Ist z. B. in einem modellbasierten Vorgehensmodell eine automatische Umsetzung der Anforderungen bedingt möglich, so steht im konventionellen Vorgehen eine manuelle Implementierung durch den Entwicklung dazwischen. Die Durchgängigkeit ist damit nicht gewährleistet.

Zum anderen setzt der modellbasierte Ansatz an dieser Stelle meist eine komplexe Verknüpfung mehrerer Modelle und deren, häufig mehrstufige Transformation in eine abstrakte User Interface Beschreibung voraus. Die Ableitung von Benutzerschnittstellen aus derart abstrakten Modellen ist noch Neuland und bedarf noch einiger Forschung [WBBG89, PE99, VLF⁺01, MFC01, PV02] (s. Abschnitt 4.3). Derart komplexe parallele Prozesse lassen sich schwer kontrollieren und gezielt im Kontext des Darstellungsprozesses manipulieren.

Weiterhin kann davon ausgegangen werden, dass sich die Anforderungen nicht in dem Maße verändern, wie dies bei den gestalterischen Merkmalen der Darstellung selbst der Fall ist. Der Entwicklungsprozess besteht hier ja in erster Linie in der Optimierung der Darstellung durch Anpassung des Designs.

Die Bearbeitung der endgültigen Darstellung, d. h. die Arbeit eines Designers, *Usability* Experten und Entwicklers (s. Abbildung 5.3), besteht also in diesem Modell darin, auf die einzelnen Prozesse zuzugreifen und diese anzupassen. Man muss hierbei unterscheiden zwischen den Prozessen, deren Eigenschaften anwendungsübergreifend konstant gehalten werden können und denen, die speziell auf einzelne Anwendungen und Benutzerschnittstellen angewandt werden.

Plattform Standards Dieser Prozess ist einheitlich über verschiedene Anwendungen. Standards können generell für Plattformen definiert werden, sie können aber auch im Sinne von *Corporate Design*, d. h. herstellerepezifischen Standards für verschiedenen Anwendungen eines Herstellers gelten. Dieser Prozess muss damit für eine Plattform nur einmal entwickelt werden und unter Umständen periodisch optimiert werden. Er könnte vom Hersteller zur Verfügung gestellt werden (s. *Device Templates* bei [Int05a]).

Spezifisches Design Dieser Prozess ist speziell auf eine Anwendung, bzw. auf einzelne User Interfaces zugeschnitten. Hier manifestieren sich die Änderungen, welche ein Designer an der Darstellung eines User Interfaces vollzieht. Dieser Prozess muss damit also für jedes User Interface und für jede Plattform angepasst werden werden.

Kontextinformationen Die Kontextinformationen werden in der Regel nicht zur Designzeit erfasst, sondern sind ein Produkt der zur Laufzeit eingesetzten Kontextsensoren. Somit ist dieser Prozess zwar hoch dynamisch, stellt aber nicht Teil des Entwicklungsprozesses dar.

Die sequenzielle Vorgehensweise ermöglicht eine gezielte Optimierung der verschiedenen Prozesse. Wichtig ist hierbei die Möglichkeit der Manipulation *bottom-up*, also basierend auf einer möglichst konkreten Darstellung. Diese Anforderung wurde in den Abschnitten 4.4.1.2 und 2.4.2 deutlich und resultierte in der Forderung einer möglichst einfachen und konkreten Bearbeitung, wünschenswerter Weise mittels graphisch-interaktiver Werkzeuge. Dies bedeutet, der Zugriff auf jede Ebene des Modells sollte aus der konkreten Darstellung und damit aus dem Detaillierten User Interface heraus möglich sein. Realisiert werden kann dies durch eine Unterdrückung oder Konstanthaltung der übrigen Prozesse. Dies ermöglicht den gezielten Zugriff auf einzelne Prozesse. Abbildung 5.6 zeigt den durchgängigen Darstellungsprozess mit Unterdrückung bzw. Konstanthaltung zur Bearbeitung einzelner Prozesse.

Abstrakte User Interface kann vom Designer angepasst werden, indem Design und Kontext Anpassung unterdrückt werden, die Darstellung wird dann einzig durch die Plattform Standards determiniert, welche konstant gehalten werden und durch die abstrakte UI Beschreibung welche dann Gegenstand der Bearbeitung ist. Das Abstrakte UI wird jedoch primär über die Anforderungen definiert werden.

User Interface Standards werden bearbeitet, indem man die nachfolgenden Prozesse der designspezifischen und kontextspezifischen Anpassung unterdrückt. Die Darstellung ist dann, ebenso wie oben nur durch die Standards determiniert, welche in diesem Fall dann auch direkt bearbeitet werden können.

Design Anpassung stellt das klassische UI-Design durch Designer oder Usability Experten dar. Hier wird nicht mehr die grundlegende Struktur manipuliert, sondern die Gestaltung im Sinne von Farbgebung, Position und anderen Eigenschaften. Dies wird durch die Unterdrückung der Kontext Anpassung und durch die Konstanthaltung der Plattform Anpassung erreicht.

5.2.2 Geordnete Entwicklungspfade

Das Prinzip der geordneten Entwicklungspfade, oder auch *ordered-path development*, wird im Kontrast zu dem Prinzip der Multi-Pfad Entwicklung (*multi-path development*) [CCT⁺03, LVM⁺04] gesehen. Die Kernprinzipien der Multi-Pfad Entwicklung wurden im Rahmen des CAMELEON Referenz Modells [CCT⁺03] definiert und basieren auf der Transformation von Modellen (s. Abbildung 5.7).

Einer der zentralen Ansätze der Multi-Pfad Entwicklung ist, wie der Name bereits sagt, das Prinzip der *multiplen Entwicklungspfade* [TCC04]. Es besagt, dass mehrere Transformationen zu einem Entwicklungspfad zusammengefasst werden können (z. B. der schrittweisen Transformation von

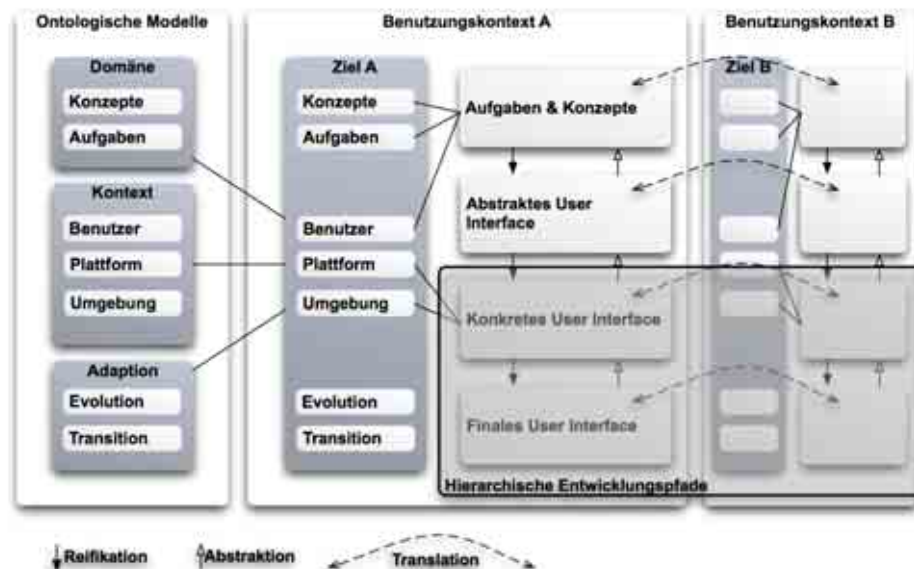


Abbildung 5.7: Einordnung des Referenzmodells der geordneten Entwicklungspfade in das CA-MELEON Referenzmodell der modellbasierten plattformübergreifenden Entwicklung [TCC04].

einem Aufgabenmodell zu einem Dialogmodell) und mehrere Pfade nebeneinander existieren können. Die Entwicklungspfade können flexibel in verschiedene Richtungen zeigen, so kann z.B. *top-down*, *bottom-up* und aus der Mitte heraus (*middle-out*) entwickelt werden. Limbourg et al [LV04] beschreiben eine Methode, wie diese regelbasierten Transformationen auf die Übergänge zwischen den Ebenen (vertikal in dem Modell) angewendet werden können.

Die Flexibilität dieses Ansatzes kann für einen erfahrenen Entwickler sehr mächtig und vorteilhaft sein. Jedoch besteht auch die Gefahr, den Entwickler zu überfordern. Entsprechend den Anforderungen aus Abschnitt 2.4 soll der hier vorgestellte Ansatz möglichst einfach gestaltet sein und die zugrundeliegenden Prozesse bei Bedarf für den Benutzer transparent gestalten. Die Unterstützung der heute geläufigen Entwicklungsmethoden scheint aufgrund der in 2.4 gesammelten Erkenntnisse ebenfalls von großer Bedeutung. Aus diesem Grunde wurde das *Konzept der geordneten Entwicklungspfade* formuliert.

Zwei zentrale Annahmen des Prinzips der geordneten Entwicklungspfade bestehen zum einen darin, dass die Bearbeitung und Design von plattformübergreifenden User Interfaces zwischen verschiedenen Rollen aufgeteilt ist (s. Abbildung 5.3), und zum anderen die Bearbeitung durch Designer und Usability Experten primär durch graphische, direkt-manipulative Werkzeuge erfolgen soll. Die Bearbeitung kann somit an verschiedenen Ebenen ansetzen und ist je nach dem auf verschiedene Aspekte des Darstellungsprozesses fokussiert. Die geordneten Entwicklungspfade betreffen hierbei in erster Linie die Bearbeitung der bereits erstellten abstrakten Spezifikation, bzw. des plattformspezifischen Designs durch den Designer oder Usability Experten.

Nach dem Prinzip der geordneten Entwicklungspfade (s. Abbildung 5.8) erfolgt die Entwicklung in hierarchisch geordneten Pfaden. Diese Pfade beginnen, wie bereits oben erklärt bei dem Abstrakten User Interface und enden bei der Darstellung. Nicht berücksichtigt wird die Erzeugung des abstrakten UI, entweder automatisch aus einem Modell oder manuell aus einer Anforderungsspezifikation.

Der dominierende Entwicklungspfad wird als der *primäre Pfad* bezeichnet. Dieser dient als Referenz. Dem primären Pfad nachgeordnet kann eine beliebige Zahl an *sekundären* Entwicklungspfaden stehen (s. Abbildung 5.9 für ein Beispiel).

Als *primärer Entwicklungspfad* wird in der Regel die Entwicklung für die *gewohnte* Plattform dienen. D. h. ein Designer, welcher bereits einige Expertise in der Gestaltung von User Interfaces für die PC-Plattform gewonnen hat, wird als primären Entwicklungspfad den Darstellungsprozess für

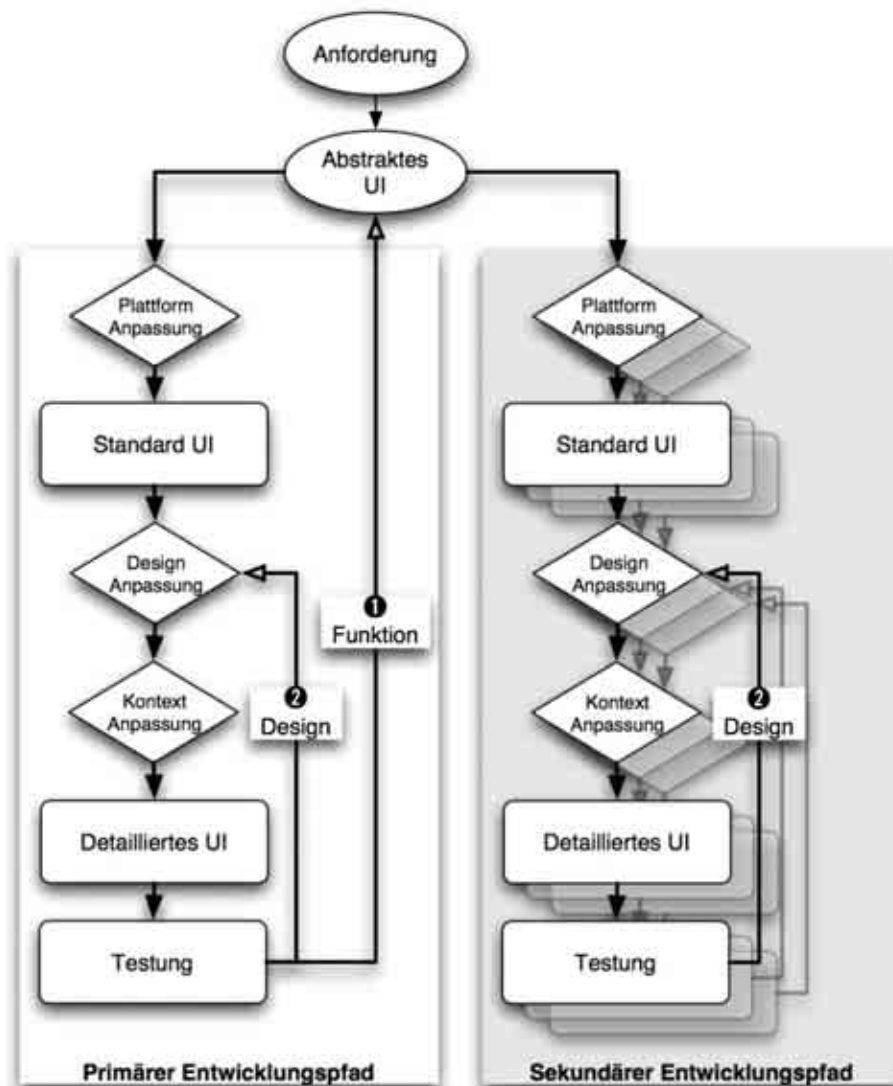


Abbildung 5.8: Darstellung der Primären und Sekundären Entwicklungspfade.

die PC-Plattform wählen. Steht ihm ein graphischer WYSIWYG User Interface Builder zur Verfügung, wird er hier also zunächst die Realisierung des Abstrakten User Interfaces für die PC-Plattform dargestellt bekommen.

In dieser vertrauten Darstellung kann der Designer nun zwei Arten von Anpassungen vornehmen: entweder die Anpassung des *plattformspezifischen Designs* (ein Element soll speziell auf dieser Plattform bestimmte Darstellungseigenschaften, z. B. Farbe, haben; s. Abbildung 5.8: (2) *Design*) oder die Anpassung der des *Abstrakten User Interfaces* (d. h. ein User Interface Element soll auf *allen* Plattformen bestimmte Darstellungseigenschaften haben; s. Abbildung 5.8: (1) *Funktion*).

Der Designer muss also definieren, wofür die letzte Änderung gelten soll, für diese Plattform, oder für alle Darstellungen gleichermaßen. Somit ist über die Referenzplattform eine Feinanpassung der vom Entwickler erzeugten abstrakten Benutzerschnittstelle durch den Designer möglich.

Die *sekundären Entwicklungspfade* bieten im Gegensatz zum primären Pfad lediglich die Möglichkeit zur Anpassung des plattformspezifischen Designs. Der Zugriff auf die Abstrakte User Interface Spezifikation ist beschränkt. Eine Veränderung der Eigenschaften eines User Interface Elements in der Darstellung einer sekundären Plattform (z. B. der Pocket PC 2002 Plattform) wird also direkt dem plattformspezifischen Designprozess zugeordnet.

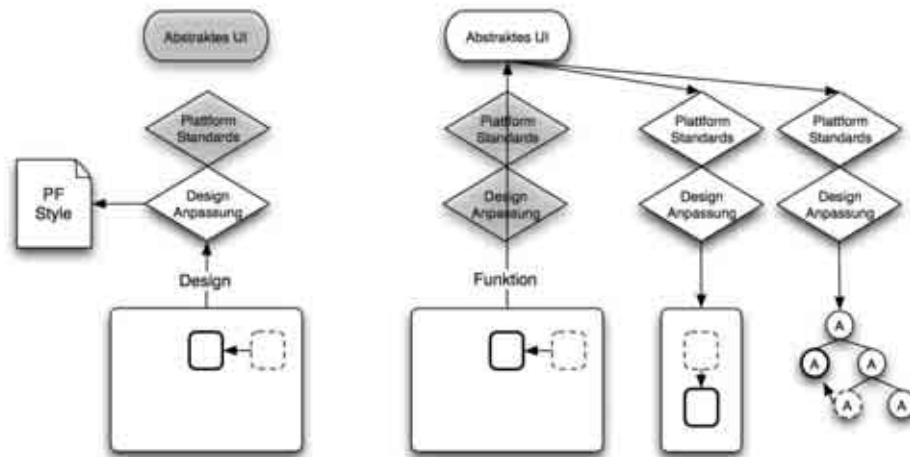


Abbildung 5.9: Ein Beispiel für die Anwendung des Prinzips der geordneten Pfade. Links wird lediglich eine plattformspezifische Designänderung vorgenommen. Rechts wird eine Änderung des Abstrakten UI vorgenommen. Diese Änderung wirkt sich auf die anderen Plattformen gleichermaßen aus.

Weswegen sollte das Design über den sekundären Pfad eingeschränkt sein? Tatsächlich gibt es mehrere Gründe, welche diese Vorgehensweise rechtfertigen und sie vorteilhaft erscheinen lassen.

Zum einen wird die *Benutzbarkeit* des Entwicklungsprozesses und die *Vorhersagbarkeit der Ergebnisse* durch die Einschränkung der Eingriffsmöglichkeiten verbessert.

Durch die Fokussierung auf eine primäre Designplattform wird es für den Entwickler einfacher, eine konsistente Vorstellung von der Anwendung zu entwickeln. Er konzentriert sich auf die Darstellung auf einer Plattform, optimiert diese und greift über diese auf die zugrundeliegenden Strukturen zu. Hierdurch werden für den Entwickler auch die kausalen Zusammenhänge zwischen abstraktem Modell und Darstellung transparenter. Der Designer lernt auf diese Weise, die Darstellungsprozesse zu verstehen und deren Ergebnisse zu antizipieren.

Bietet das System die Möglichkeit, an mehreren Schrauben gleichzeitig zu drehen, entsteht sehr schnell der Effekt, dass sich Änderungen *aufschaukeln* das Verhalten des Systems *unvorhersagbar* wird und Effekte nicht mehr zuzuordnen sind. So kann dann nicht mehr festgestellt werden ob eine Veränderung der Darstellung Ergebnis der Veränderung der abstrakten Struktur oder eine Wechselwirkung mit der Plattformspezifischen Darstellung ist.

Weiterhin dient die Einschränkung der *Konsistenz des Designs* und stellt eine Anwendung des *Transformationsparadigmas* (s. u.) dar. Aufgrund der Tatsache, dass der Entwickler nun primär in einer Plattform denkt und gestaltet, kann er für diese ein konsistentes konzeptuelles Modell anhand der Darstellung entwickeln. Das System erzeugt die Darstellungen auf den anderen Plattformen weitestgehend automatisch. Der Designer erhält eine lediglich evaluierende und korrigierende Rolle. Der Entwicklungsprozess ähnelt somit dem Wahrnehmungsprozess des Benutzers und erleichtert somit dem Entwickler, diesen nachzuempfinden.

Zuguterletzt stellt eine Einschränkung der Entwicklungspfade eine erhebliche *Reduzierung des Implementierungsaufwandes* für das Entwicklungssystem dar. Es muss eine Umkehrung der Plattform-Standard Darstellungsprozesse nur für eine oder wenige Plattformen realisiert werden, da die sekundären Plattformen nur Anpassungen auf der plattformspezifischen Darstellungs-Schicht ermöglichen.

5.2.3 Das Transformationsparadigma

5.2.3.1 Hintergrund

Das *Transformationsparadigma* stellt eine der wichtigsten Annahmen des Referenzmodells dar.

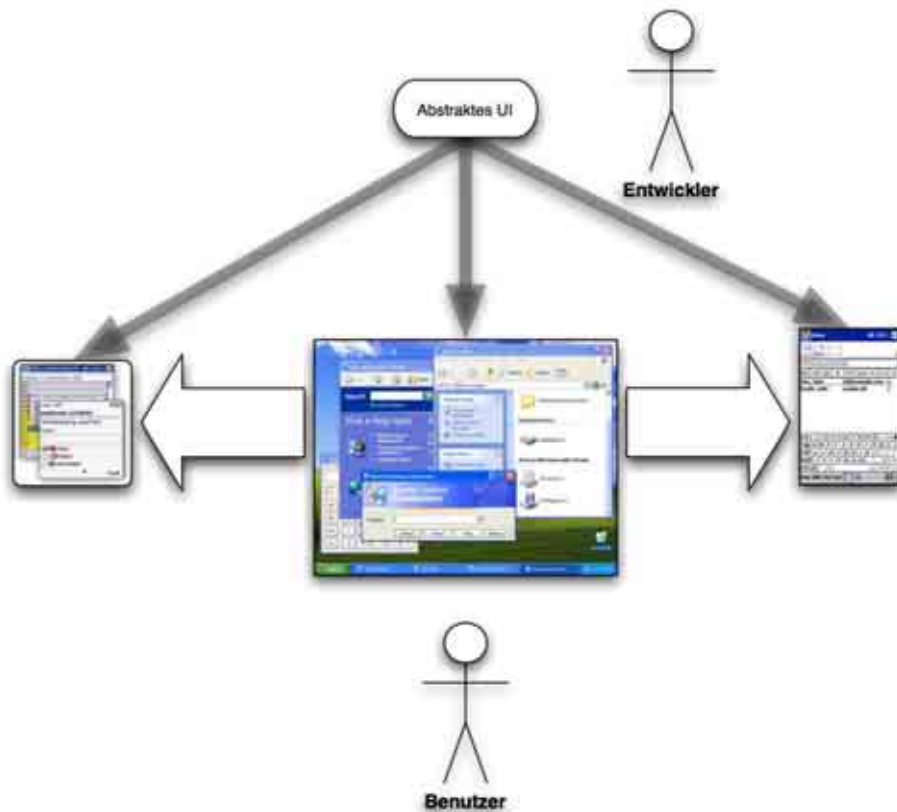


Abbildung 5.10: Wahrnehmung des plattformübergreifenden Systems aus Sicht des Benutzers vs. Entwicklers.

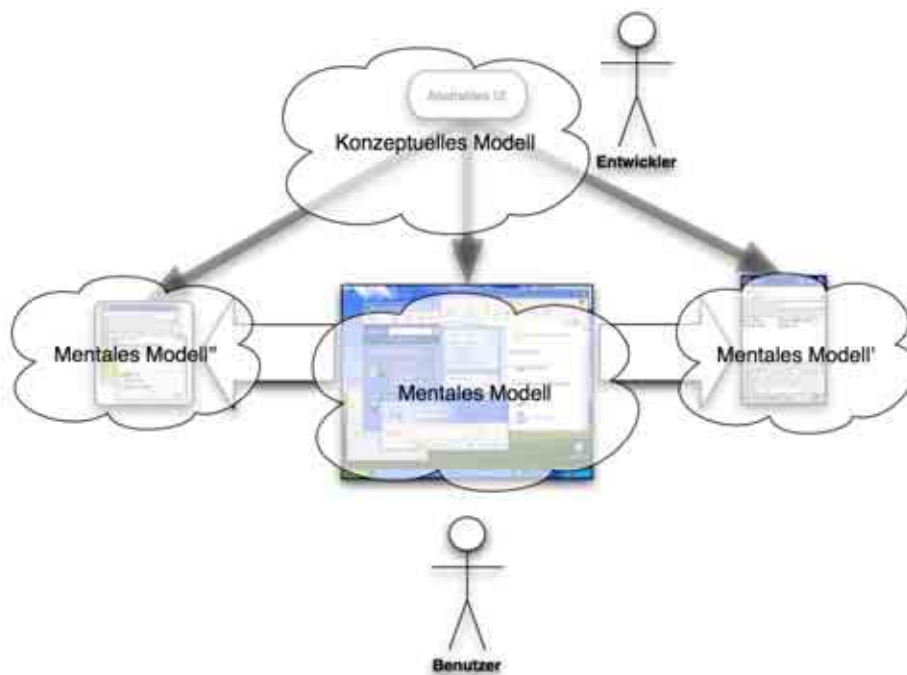


Abbildung 5.11: Modellbildung beim Entwickler und beim Benutzer.

Die Verwendung des Begriffs ‚Transformation‘ sollte hierbei mit Bedacht geschehen, da sich dieser bereits in verschiedenen Kontexten der Entwicklung von Benutzerschnittstellen findet und hier recht uneinheitlich verwendet wurde.

So beschreibt Agresti [Agr86] nach [VS95] die Generierung der operationalen Spezifikation (d. h. des Anwendungscode der Darstellung) als eine Transformation, die sich aus einer ursprünglichen Spezifikation ableitet. Der Transformationsprozess kann darauf als dokumentierter Erzeugungsprozess (*formal development record*) verstanden werden.

Limbourg [LV04] beschreibt einen Ansatz, wie mit Hilfe von Graphtransformationen Modelle verschiedener Abstraktionsebenen transformiert, konkretisiert und so auf verschiedene Darstellungs- oder Kontextparameter angepasst werden können. Die Transformationsannahme liegt hierbei dem Kernprozess des UML Darstellungsprozesses zugrunde. Die Transformation ist hierbei relativ unabhängig von der inhaltlichen Bedeutung der zu transformierenden hierarchischen Datenstrukturen und kann somit auf alle Ebenen der Reifikation eingesetzt werden.

Transformationsregeln werden als formale Regeln in dem Transformationssystem in einer definierten Syntax abgelegt und während des Transformationsprozesses angewandt [LVM⁺04, SLV⁺05]. Transformation wird hierbei also in erster Linie im Prozess der Konkretisierung eingesetzt, wenn ein Modell aus einer höheren Abstraktionsebene konkretisiert werden soll, nach dem CAMELEON Modell als *Reifikation* bezeichnet [CCT⁺03]. Ansätze der Translation (Anpassung auf derselben Abstraktionsebene) oder Abstraktion wurden noch nicht vorgestellt.

Transformationen werden in der modellbasierten Entwicklung von Benutzerschnittstellen bereits seit langem eingesetzt, um zusätzlich zu der Generierung der Darstellung, eine Anpassung derselben zu ermöglichen [Kov94, Kov93]. Kovacevic beschreibt die Verwendung von Transformationen bei der Anpassungen von Benutzerschnittstellen als komplementär zu dem kompositionalen Prozess der Erstellung im TACTICS-System bzw. dem UIMS UIDE [FGK88]. Transformationen ermöglichen in diesen System die Anpassung der Darstellung auf globalem oder elementarem Niveau, wobei TACTICS eine feinere Transformationsanpassung ermöglicht.

Die existierenden Ansätze der Verwendung von Transformationen sind in erster Linie technisch motiviert, basierend auf der verwendeten Datenrepräsentation. In erster Linie stehen hierbei die technischen generativen Prozesse im Vordergrund ohne hierbei zu berücksichtigen ob dies für den Benutzer relevant und von Nutzen ist. Im Rahmen der benutzerzentrierten Vorgehensweise erfolgt hier eine Neubetrachtung des Transformationalen Ansatzes und eine theoretische und empirische Anleitung der Relevanz solcher Methoden für den Entwicklungsprozess.

5.2.3.2 Neubetrachtung

Bei der transformationalen Annahme wird davon ausgegangen, dass der Benutzer den Wechsel zwischen Plattformen als *Transformation* der Anwendung wahrnimmt. Die beim Übergang von einer Plattform zur anderen wahrgenommene Veränderung der Anwendungsoberfläche wird als *Abbildung* einer Oberfläche auf die andere wahrgenommen ($O \rightarrow O'$).

Weiterhin wird bei dem Transformationsparadigma davon ausgegangen, dass der Benutzer diese Abbildung nicht als symmetrisch empfindet, sondern eine Plattform, in der Regel die am häufigsten genutzte, dominiert, während weniger vertraute Plattformen unterdrückt werden (s. auch *Prinzip der geordneten Pfade*). Die Anwendung auf einer sekundären (unterdrückten) Plattform wird als Variante der Anwendung auf der primären (dominanten) Plattform wahrgenommen.

Abbildung 5.11 verdeutlicht die Transformationsannahme: während der Entwickler die verschiedenen Darstellungen als Resultat von Abbildungen aus demselben abstrakten Modell betrachtet, stellt sich dem Benutzer lediglich das Ergebnis des Darstellungsprozesses als Phänomen oder Systembild [Nor83, Nor88] dar.

Die Transformationsannahme stützt sich unter anderem auf die Ergebnisse der explorativen Benutzerstudie, welche in Abschnitt 2.2 beschrieben wurde. Es zeigte sich hier deutlich dass versucht wurde, Regeln und Prozesse von einer Plattform auf die andere zu übertragen („Wie ging das vorhin?“). Anwender versuchten die Anwendung mittels des Wissen, welches sie auf der einen Plattform erworben hatten, auf der anderen Plattform zu verstehen und zu bedienen. In der Studie wurden

Fehler, welche aus einer fehlerhaften Übertragung zwischen den Anwendungen auf verschiedenen Plattformen resultierten, als regelbasierte Fehler bezeichnet [ER86, Rea90]. Es wurde davon ausgegangen, dass die Regeln, welche auf der ersten Plattform gebildet wurden, auf die zweite Plattform übertragen wurden und dort zu fehlerhafter Bedienung führten.

Singley und Anderson [SA89] zeigen in ihrem Buch *Transfer of Cognitive Skill* auf, wie der Transfer von Wissen zwischen verschiedenen Anwendungen erfolgt. Sie weisen nach, dass bei der Bedienung von verschiedenen Arten von Texteditoren die Expertise auf eine neue Anwendung übertragen werden kann, wenn diese sich ausreichend ähneln. In Abschnitt 3.1 wird die Psychologie des *Wis-senstransfers* und die Arbeiten von Singley und Anderson ausführlicher beschrieben. Sie beschränkt sich jedoch in erster Linie auf den Transfer von Wissen auf derselben Plattform. Die Benutzerstudie weist darauf hin, dass dieselben Prozesse auch für den Transfer zwischen Plattformen wirken können. In Abschnitt 6.2 werden Methoden der Anpassung diskutiert und die Transformationsannahme empirisch überprüft werden.

Betrachten wir die klassische Erkenntnisse zu der Bildung mentaler Modelle in der Literatur [Nor83, Nor88, JL83, VPM03], so wird angenommen, dass der Benutzer dieses aus dem Systembild generiert, oder im Sinne von Johnson-Laird [JL83] die propositionalen Repräsentationen, welche er aus dem Systembild gewinnt, mittels mentaler Semantik in das mentale Modell zu integrieren versucht (s. Abschnitt 3.2). Nach Norman [Nor83] ermöglicht es ein gelungenes Systembild, welches das konzeptuelle Modell des Systems (also die durch den Entwickler angestrebte Funktionalität) in geeigneter Weise repräsentiert, es dem Benutzer, ein konsistentes und funktionales mentales Modell aufzubauen. Ein solches Modell wird als wichtige Voraussetzung für den erfolgreichen Umgang mit dem System angesehen.

Wie kann nun bei einem Wechsel zwischen Plattformen ein mentales Modell gebildet werden? Zum einen geht die Transformationsannahme davon aus, dass der Benutzer sein Wissen von einer Plattform auf die andere zu transferieren versucht. Die Anpassung des Wissens auf die neue Plattform erfolgt hierbei schrittweise und auf Basis von Hypothesentestung. Es wird also zunächst versucht, bestehende Strategien der ersten Plattform anzuwenden. Schlagen diese fehl, dann werden sukzessive weitere Strategien erprobt bis die richtige Strategie gefunden ist. Basierend hierauf wird eine neue mentale Repräsentation für das zweite System gebildet.

In der Studie in Abschnitt 2.2 konnte beobachtet werden, wie Benutzer des Pocket PC zunächst, wie auf der PC Plattform Funktionen im Kontextmenü suchten, sie dort nicht fanden, nach geeigneten Hinweisen in Listen oder anderen Interaktoren suchten und zuletzt die Menüleiste untersuchten. Sukzessive wurden Hypothesen abgearbeitet und verworfen. Der Vorgehensweise entsprach hierbei der Vorgehensweise auf der zunächst benutzten Plattform und führte umso schneller zum Erfolg, je größer die Übereinstimmung war.

Die Transformationsannahme impliziert nun die Schlussfolgerung, dass, sollte die Abbildung zwischen den Plattformen regulären und konsistenten Gesetzen folgen, die Bildung eines neuen, separaten mentalen Modells nicht oder nur eingeschränkt notwendig ist [Nie89, Nie93]. Der Benutzer kann bei ausreichender Nähe zwischen den Varianten das neue Interface aus dem alten ableiten. *Er wendet hierzu eine mentale Transformation an.*

Diese Annahme leitet sich in erster Linie aus der Theorie Johnson-Lairds [JL83] und der Struktur der menschlichen Informationsverarbeitung ab (s. hierzu 3.2). Kann das neue mentale Modell in das existierende integriert werden, wird der Benutzer versuchen, dies zu nutzen. Erst bei einer zu großen *Distanz* wird dieser Aufwand zu groß und ein neues Modell wird angelegt. Erfahrung und Wissen von der einen Plattform kann nun nur schwer und unter großem Transferaufwand übertragen werden. Der Wechsel gestaltet sich in solchen Fällen als „extrem schmerzhaft“ (so ein Teilnehmer in einer Untersuchung).

Die *zweite Annahme* des Transformationsparadigmas beinhaltet die Unterscheidung zwischen dominierender oder primärer und unterdrückter oder sekundärer Plattformen. Die primäre Plattform ist im heutigen Arbeits- oder Nutzungskontext in den meisten Fällen der Desktop PC mit *Microsoft Windows* oder *Apple Macintosh OS*. Die Erfahrung im Umgang mit der primären Plattform prägt in großem Maße den Umgang des Benutzers mit Computersystemen. Interaktionsroutinen werden durch häufiges Üben verfestigt und zu hoch automatisierten Fertigkeiten verdichtet.



Abbildung 5.12: Zwei Entwicklungsstrategien.

Sowohl die Eingabe (Tastaturbefehle, Nutzung der rechten Maustaste, Drag & Drop, etc.) als auch die Wahrnehmung und Evaluation (Interaktionselemente, Metaphern wie der Papierkorb, Fenster, etc.) sind in hohem Maße auf die Metaphern und Möglichkeiten der dominierenden Plattform ausgerichtet. Beim Wechsel zwischen den Plattformen wird dieses Wissen nicht einfach über Bord geworfen, sondern bleibt als Basiswissen verfügbar. Die Klasse der fertigkeitsbasierten Fehler, wie sie in Abschnitt 2.2 beschrieben werden, machen diesen Effekt deutlich. In der Studie konnte beispielsweise beobachtet werden, dass Benutzer, die nicht vorwiegend auf der PC-Plattform arbeiteten, deutlich seltener versuchten, das Kontextmenü des Pocket PC zu nutzen als PC-Nutzer. Aufgrund der geringen Anzahl von Versuchspersonen können hier allerdings keine quantitativen Aussagen gemacht werden. Gilroy und Harrison schlagen vor, den Interaktionsstil einer Plattform als Faktor der Anpassung zu nutzen [GH04].

Die Gültigkeit der Transformationsannahme ist theoretisch begrenzt durch die Distanz zwischen den Darstellungen auf verschiedenen Plattformen. Sind zwei Darstellungen derselben Anwendung auf zwei Plattformen noch als Varianten derselben Anwendung wahrnehmbar, besitzt das Transformationsparadigma noch Gültigkeit. Differieren die Varianten zu sehr, dann ist eine Ähnlichkeit nur noch sehr schwer wahrzunehmen, die Varianten werden mit höherer Wahrscheinlichkeit als verschiedenen Anwendungen wahrgenommen werden. Feature-basierte Ansätze wie das *Kontrastmodell* von Tversky [Tve77] könnten hierfür als Erklärungsgrundlage dienen.

Aus der Transformationsannahme lässt sich die Forderung nach einer Unterstützung des Entwicklers bei der konsistenten Gestaltung plattformunabhängiger Anwendungen ableiten. Die Entwicklungsplattform sollte dem Entwickler die Möglichkeit bieten, das gewählte Design auf einer Plattform mit den abgeleiteten Darstellungen auf den anderen Plattformen hinsichtlich ihrer Ähnlichkeit, bzw. Konsistenz zu vergleichen.

Wendet der Designer also eine Anpassung an, dann stehen ihm zwei Vorgehensweisen zur Verfügung, die beide in Abbildung 5.12 verdeutlicht werden.

- a.) Der Designer wendet die Änderung auf die allgemeine Beschreibung auf der abstrakten Ebene an. Damit verändert sich auch die Darstellung auf den anderen Plattformen. Die Plattformen werden synchronisiert.
- b.) Der Designer beschließt, dass diese Änderung eine plattformspezifische Optimierung darstellt, und deswegen nicht auf die anderen Plattformen angewandt werden soll. Er speichert diese Änderungen in einem Differenzierungsteil in der plattformspezifischen Beschreibung, diese

werden bei der Darstellung nur auf die entsprechenden Plattformklassen angewandt. Die Plattformen werden differenziert.

Die Konsistenz der verschiedenen Versionen wird automatisch vom System verglichen. Der Entwickler erhält permanent Feedback über die Übereinstimmung der Oberflächen und kann diese daraufhin selbst oder automatisiert optimieren. Die Konsistenzmessung kann gleichermaßen auch bei der Entwicklung der Abbildungsregeln genutzt werden.

Der Entwicklungsprozess wird als ein Prozess dargestellt, welcher aus zwei, sich gegenseitig kontrollierenden Prozessen besteht: die *plattformspezifische Optimierung*, welche der Designer aufgrund seines Wissens auf die automatischen Abbildungsregeln aufsetzt, und die *plattformübergreifende Harmonisierung* durch die automatische Verfolgung der Übereinstimmung, bzw. Konsistenz der Darstellung auf den verschiedenen Plattformen. Der Designer wird also bei der Optimierung dahingehend unterstützt, dass er die permanente Kontrolle der Konsequenzen seiner Eingriffe auf die plattformübergreifende Darstellung erhält.

Diese Vorgehensweise ermöglicht die direkt-manipulative Bearbeitung der Darstellung plattformübergreifender Oberflächen direkt in der Darstellung selbst. Der Designer muss nicht selbst sämtliche Darstellungsvarianten verfolgen, sondern erhält eine Kontrolle über das Konsistenzmaß, welches die Übereinstimmung reflektiert.

Die Entwicklung eines Konsistenzmaßes, welches die Überprüfung der Übereinstimmung der Darstellung zwischen den Plattformen ermöglicht, stellt eine wichtige Voraussetzung für dieses Vorgehen dar. Das Konzept eines plattformübergreifenden Konsistenzmaßes, welches auf das Transformationsparadigma aufbaut wird in Abschnitt 5.4 diskutiert. Die Umsetzung eines solchen Konsistenzmaßes und dessen Validierung wird in Abschnitt 5.4 ausführlicher behandelt werden.

5.3 Transformationsstrategien

Die im *Konzept plattformübergreifender Entwicklung* beschriebene Transformationsannahme geht davon aus, dass der Benutzer beim Wechsel zwischen verschiedenen plattform- oder kontextspezifischen Instanzen derselben Anwendung, diesen als Transformation in der Darstellung wahrnimmt.

Die Überlegung war weiterhin, dass eine regelmäßige und voraussagbare Anwendung von Transformationen, es dem Benutzer erleichtern soll die Abbildung zwischen Instanzen nachzuvollziehen, bzw. während der Interaktion zu antizipieren. Dieselbe Transformation, die von Instanz *A* auf Instanz *B* vollzogen wurde, kann der Benutzer mental auf sein Anwendungsmodell anwenden und damit existierendes Wissen weiterverwenden.

In diesem Abschnitt sollen nun exemplarisch am Beispiel graphischer Benutzerschnittstellen verschiedene Strategien zur Anpassung der Darstellung an andere Plattformen vorgestellt werden. Diese Strategien werden im Sinne der Transformationsannahme als Transformationen beschrieben und geordnet und klassifiziert.

Woran muss die Darstellung nun angepasst werden? Nach Weiss [Wei02] sind die wichtigsten Eigenschaften von informationstechnischen Geräten, deren Nutzungsgrund und Kontext, auf die ein Gerät durch Design, Form und Abmessungen, Interaktionsmittel, Display und Speicher optimiert werden sollte. Anpassung sollte diese Parameter also stets als Grundlage für Transformationen nutzen. Im Falle graphischer Benutzerschnittstellen ist die Anpassung an Darstellungsgrößen und Eingabegeräte die häufigste Form dieses Problems, wenn eine Anwendung für Desktop PCs, Handheld Devices und Smartphones zugleich entwickelt werden soll. Auf all diesen Geräten sind prinzipiell ähnliche Interaktionselemente vorhanden. Gleichzeitig variieren diese Geräte stark in Displaygröße und Ein- und Ausgabegeräten.

5.3.1 Graphische Anpassungsmechanismen

Bei der Anpassung von Anwendungen für verschiedene Endgeräte, wird meist die Anpassung auf die Größe des Displays im Vordergrund stehen. Automatisierte Darstellungsprozesse, wie sie z. B.

von modellbasierten Ansätzen unterstützt werden, setzen diese Strategien zum Teil um. Dennoch bedarf es bei der Entwicklung von geeigneten Anpassungsmechanismen noch einiger Forschung bis diese zufriedenstellende, konsistente und benutzergerechte Darstellungen erzeugen. Ausgehend von der Transformationsannahme ist es hierbei vor allem wichtig, dass diese den Benutzer bei der Entwicklung eines konsistenten mentalen Bildes [EK93] der Anwendung, also eines mentalen Modells [GS83, JL83, Nie86, RH04a] unterstützen.

Unter den Dimensionen, an die eine Benutzerschnittstelle angepasst werden muss, ist heute neben der Art des verfügbaren Eingabegeräts das Ausgabegerät, und hier insbesondere die Größe des Bildschirms die wichtigste Dimension. Die Anpassung an die Displaygröße und Auflösung betrifft in erster Linie die oberflächlichen Eigenschaften eines Interfaces, wie die Form der Elemente, deren räumliche Anordnung, welche die Fähigkeit des Benutzers, Gemeinsamkeiten zwischen verschiedenen Versionen eines Interfaces zu erkennen und somit vorhandenes Wissen über die Anwendung wiederzuverwenden wesentlich beeinflussen [DK04].

Vanderdonckt *et al.* [VLF⁺01] unterscheiden bei ihrer Untersuchung zwischen Bildschirmauflösung (in Pixel) und Bildschirmgröße (in cm). Sie schlagen folgende Anpassungsstrategien für Geräte mit niedriger Auflösung vor: Verkleinerung, Ersetzen von Elementen durch kleinere Alternativen und die Umordnung und Unterteilung von Dialogen in Präsentationsgruppen auf Basis von Aufgabenmodellen. Um diesen Ansatz systematisch fortzuführen, wird ich an dieser Stelle vorgeschlagen, eine Klassifikation von möglichen Transformationsstrategien anhand der Kriterien *Form*, *räumliche Anordnung* und *Anzahl*, in Anlehnung an die Überlegungen von Denis und Karsenty [DK04] einzuführen (siehe hierzu auch die Diskussion in Abschnitt 5.4) [Ric05b].

Strategie	Kriterium		
	Form	Anordnung	Anzahl
Erhaltende Transformationen	×	×	×
Neuanordnung	×	—	×
Reduktion	×	—	—
Vereinfachung	—	×	×

Tabelle 5.1: Klassifikation der Transformationsstrategien

Es wird angenommen, dass jede Anpassung einer Benutzerschnittstelle auf ein neues Gerät auf diesen Transformationen aufbaut. Das bedeutet, dass eine Anpassung stets eine Verkettung von einzelnen Transformationen verwendet, welche sich wiederum in diese Klassifikation einordnen lassen. Zweck dieser Klassifikation ist daher in erster Linie die Anpassungsstrategien einordnen zu können und den Vorgang der Anpassung zu systematisieren.

Weiterhin wird in Abschnitt 5.2.3 beschrieben, wie die *Distanz* zwischen verschiedenen Varianten einer Benutzerschnittstelle Einfluss auf die Bildung des Mentalen Modells und auf die Wiederverwendbarkeit von Wissen nimmt. In der Folge soll die Ausgangsdarstellung des Interfaces *Original* und die durch Transformation abgeleitete Version *Variante* genannt werden. Die *Distanz* kann definiert werden, als der *kognitive Aufwand*, der nötig ist, eine Variante in die andere zu überführen. Auf der Präsentationsebene würde dies unter Annahme der Transformationshypothese bedeuten, dass die Art und Kombination der verwendeten Transformationen die Distanz zwischen Varianten einer Benutzerschnittstelle bestimmt.

Folgende Annahmen lassen sich hieraus ableiten:

Konsistente Verwendung Die einheitliche Anwendung einer bestimmten Transformationsstrategie erleichtert die Anpassung des mentalen Modells. Der Benutzer kann mit ihrer Hilfe die Darstellung der Variante antizipieren. Er muss lediglich die Transformation (bzw. eine mentale Repräsentation derselben) auf sein mentales Modell anwenden und nicht die einzelnen Aspekte der Modifikation behalten.

Kognitives Gewicht Jede Transformationsstrategie hat ein spezifisches kognitives Gewicht. Das bedeutet, die mentale Umsetzung einer Transformation belegt kognitive Kapazität. Der kognitive Verarbeitungsaufwand unterscheidet sich hierbei zwischen den Transformationsstrategien. Bei der Umsetzung sollten Strategien mit geringem Gewicht bevorzugt werden. Besteht eine Transformation aus einer Verkettung von einzelnen Transformationen akkumulieren sich die Einzelaufwände.

Diese Erkenntnisse lassen bei der Entwicklung plattformübergreifender Benutzerschnittstellen umsetzen. Zum einen können Transformationsstrategien als *Baukastenelemente* in einer Entwicklungsumgebung verfügbar gemacht werden. Designer können dann für bestimmte Anordnungen von Elementen konsistent Transformationen zuweisen (z. B. wenn vier Schaltflächen in einer horizontalen Reihe stehen, dann versuche stets, diese um 90° zu rotieren).

Die *Nähe* zweier Varianten kann aufgrund des kognitiven spezifischen Gewichts der einzelnen verwendeten Transformationen geschätzt werden. Die Schätzung dieser Gewichte ist dafür notwendig.

Transformationsmuster können in der Art von Entwurfsmustern [AIS⁺77] abstrahiert, geordnet und wiederverwendet werden. Konkrete Vorschläge und Beispiele der Umsetzung werden in Abschnitt 6.2 diskutiert. Im folgenden Abschnitt sollen zunächst Beispiele für Transformationsstrategien gesammelt und beschrieben werden.

5.3.2 Erhaltende Transformationen

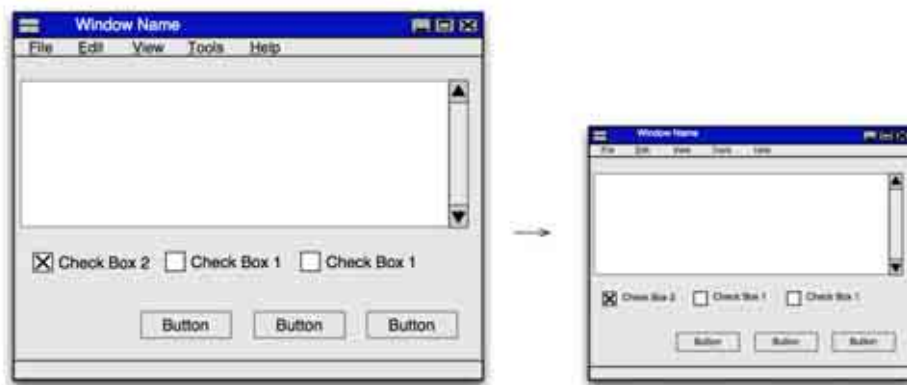


Abbildung 5.13: Erhaltende Transformationen: Skalierung

Erhaltende Transformationen erhalten alle drei Kriterien der Transformation konstant, eine Bedingung, welche lediglich durch eine Skalierung erfüllt ist. Eine Skalierung (Vergrößerung / Verkleinerung) bewahrt das relative Layout und die Anzahl der Elemente in dem Interface.

5.3.2.1 Skalierung

Um Anzahl und Anordnung von Interface-Elementen über den Transformationsprozess möglichst konstant zu halten, kann ein *Skalierungsfaktor* angewandt werden. Diese Strategie wird insbesondere bei der Übersetzung von Anwendungen von Desktop PC auf Subnotebooks und leistungsfähige Kleinstcomputer (z. B. Fujitsu Stylus) angewandt.

Problem dieser Strategie ist augenscheinlich die begrenzte Anwendbarkeit: der Verkleinerung ist durch die Lesbarkeit, der Vergrößerung, durch die Ästhetik, sowie durch die Übersichtlichkeit Grenzen gesetzt. Perlin *et al.* [PM99] schlagen eine adaptive Skalierungsstrategie (*Zooming*) der aktiven Teile des UI vor. Bedersen *et al.* schlagen mit der DATELENS [BCCR04] eine Anwendung des *Fisheye*-Ansatzes [Bed00] für eine Kalenderanwendung auf Pocket PC-Geräten vor. Hierbei wird der Teil eines Fensters, einer Liste oder Datentabelle vergrößert dargestellt.

5.3.3 Neuordnung

Die *Neuordnung* einer Oberfläche bedeutet, seine räumliche Anordnung aufzubrechen. Dies kann durch eine Rotation, eine Linearisierung oder eine Segmentierung des Interfaces erfolgen. All diese Strategien haben zum Ziel die relative Position der Elemente an die Abmessungen des Bildschirms anzupassen. Bei der Neuordnung oder Umgruppierung werden einzelne Elemente oder Elementgruppen neu positioniert. Sie können hierbei sowohl die relative Position zu anderen Elementen als auch den gesamten Kontext wechseln. Beispiel hierfür ist die Umordnung von Menü-Elementen zwischen oder in neue Menüs [SC02].

5.3.3.1 Translation

Die Translation eines Elementes besteht aus der Veränderung der Position und stellt damit die allgemeinste Form der Neuordnung dar.

5.3.3.2 Rotation

Die *Rotation* der gesamten Oberfläche oder von Teilen, dient insbesondere der Anpassung der Orientierung. So haben z. B. PC Displays eine horizontale Ausrichtung (Faktor $640pt \times 480pt$), während Pocket PC Displays hingegen vielmehr vertikal (Faktor $240pt \times 320pt$) ausgerichtet sind.

Man kann hierbei zwischen einer Rotation der Anker (d. h. eine Umordnung der Position der Elemente, nicht aber deren vertikaler Ausrichtung, also die Schrift usw. bleiben in der vorherigen Ausrichtung) und einer vollständigen Rotation (d. h. auch die Ausrichtung der Elemente wird rotiert) unterscheiden. Eine Rotation wird in der Regel aufgrund der orthogonalen Organisation graphischer Oberflächen in vollen 90 Grad Schritten erfolgen.

5.3.3.3 Spiegelung

Die *Spiegelung* einer Benutzerschnittstelle kann generell entlang einer Achse (in der Regel vertikal oder horizontal) bzw. an einem Punkt (entspricht einer Rotation) ausgerichtet sein.

Insbesondere eine axiale Spiegelung kann dann sinnvoll sein, wenn die Interaktionsrichtung [Sea93] sich zwischen Geräten unterscheidet. Im Falle der Pocket PC Plattform ist beispielsweise sowohl Menü als auch Symbolleiste am unteren Rand des Bildschirms angeordnet um eine Verdeckung der Oberfläche durch die Hand des Benutzers zu verhindern. Dadurch kann sich eine generelle Umorientierung ergeben, die sich in eine Spiegelung abbilden lässt.

Ein anderes Beispiel ist die Anpassung auf die linkshändige Benutzung. Meist sind die Elemente so angeordnet, dass diese vom rechten Rand optimal erreichbar sind um den rechtshändigen Benutzer zu unterstützen. Eine Spiegelung kann dies einfach für den Linkshänder anpassen.

5.3.3.4 Linearisierung

Die *Linearisierung* ist eine kontinuierliche Art der Umordnung anhand eines Ordnungskriteriums. Diese Umformung eignet sich insbesondere zur Anpassung an eine Sprachausgabe, da die Sprache primär entlang einer Dimension dargestellt wird. Luyten schlägt zum Beispiel die Linearisierung eines *Grid Layouts* vor, um Platz zu sparen [LVC02].

5.3.3.5 Segmentierung

Übersteigt die Menge an Informationen innerhalb eines Dialoges die Darstellungsmöglichkeiten des Endgeräts, dann muss dieser Dialog in Unterdialoge aufgespalten werden, welche Teile diese Dialoges enthalten. Die Segmentierung ist ebenfalls ein Sonderfall der Umordnung, anhand zweier oder

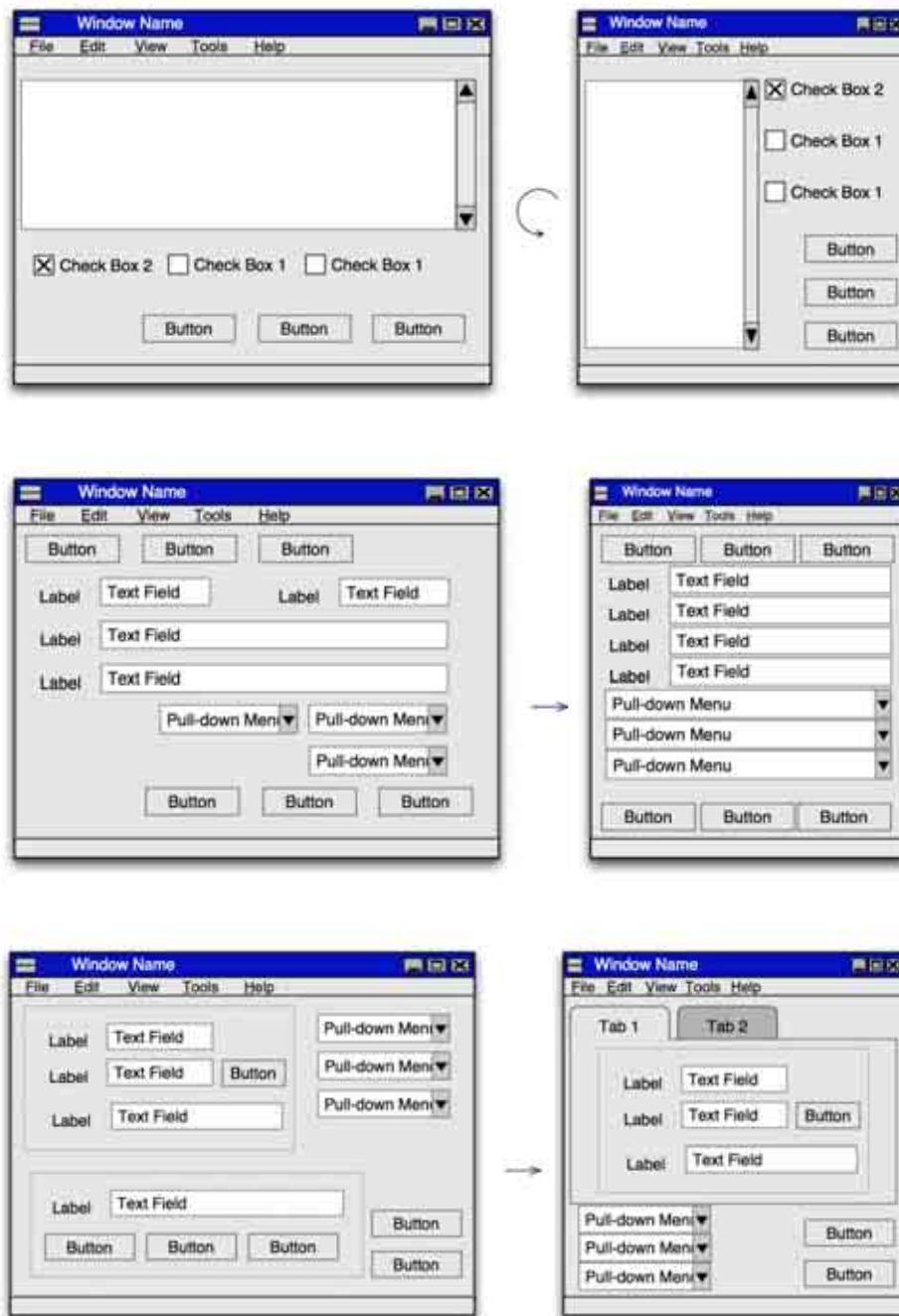


Abbildung 5.14: Transformationen der Neuordnung: Rotation, Linearisierung, Segmentierung

mehr Ordnungskriterien, wobei das erste Kriterium die Zuordnung zu einem Unterdialog, und die weiteren Kriterien die Anordnung innerhalb des Unterdialogs festlegen. Die Segmentierung lässt sich durch eine Zusammenlegung wieder aufheben [VFO01].

5.3.4 Reduktion

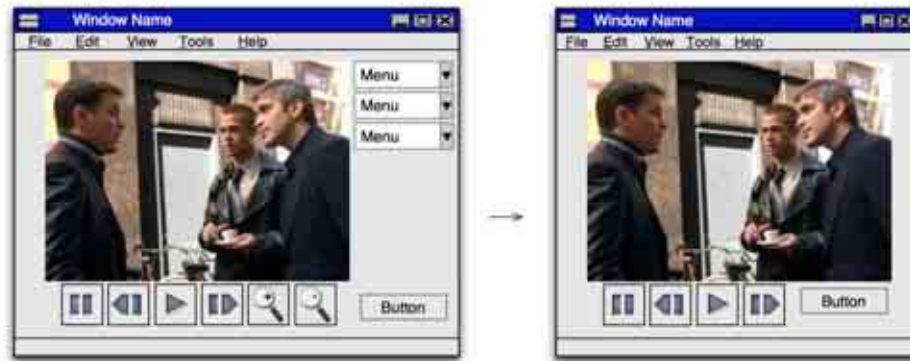


Abbildung 5.15: Strategien der Reduktion: Auslassung.

Bei einer *Reduktion* wird die Anzahl der Elemente verkleinert. Hierdurch verändert sich ebenfalls der relative Zusammenhang zwischen den Elementen. Die Auslassung von Elementen kann durch Aufgabenstrukturen oder durch Konzepte wie Detailschichten oder Priorisierungen umgesetzt werden [KP02].

5.3.4.1 Auslassung

Die *Auslassung* lässt sich ebenfalls als ein Sonderfall der Umordnung betrachten. Bei ihr werden einzelne Elemente oder Elementgruppen ausgeblendet, z. B. um den Umfang der Oberfläche an die beschränkten Möglichkeiten eines Endgeräts anzupassen [KP02, Mye04].

5.3.5 Vereinfachung

Eine *Vereinfachung* kann durch eine Vereinfachung oder Reduzierung des Inhaltes oder durch die Vereinfachung der Form von Elementen umgesetzt werden. So kann zum Beispiel der textuelle Inhalt einer Beschriftung durch eine kürzere Version des Textes ersetzt werden oder ein platzaufwändiges Element durch eine kompaktere Version ersetzt werden [VLF⁺01].

5.3.5.1 Reduzierung des Inhaltes

Eine Reduzierung der Informationsmenge kann analog zur graphischen Skalierung durch Abkürzung, sozusagen eine *syntaktische Skalierung*, realisiert werden. Durch eine Kürzung von Beschriftungen und Texten kann der zur Verfügung stehende Platz auf dem Bildschirm eines Gerätes effizienter genutzt werden. So kann zum Beispiel auf einem Desktop PC die Beschriftung einer Schaltfläche *Übernehmen* lauten, während diese auf einem Pocket PC auf ein knapperes *OK* reduziert wird [GVRV02, EVP00].

Die untere Grenze dieser Transformation ist, ähnlich wie bei der Skalierung die Reduzierung bis zur Unverständlichkeit für den Benutzer. Der Benutzer muss selbstverständlich noch ausreichend Information erhalten, um die Bedeutung zu verstehen. Nach oben hin ist die Grenze durch eine sinnvolle, noch verarbeitbare Menge begrenzt. Die Beschriftung einer Schaltfläche mit mehreren Sätzen wäre der Bearbeitungsgeschwindigkeit vermutlich abträglich.

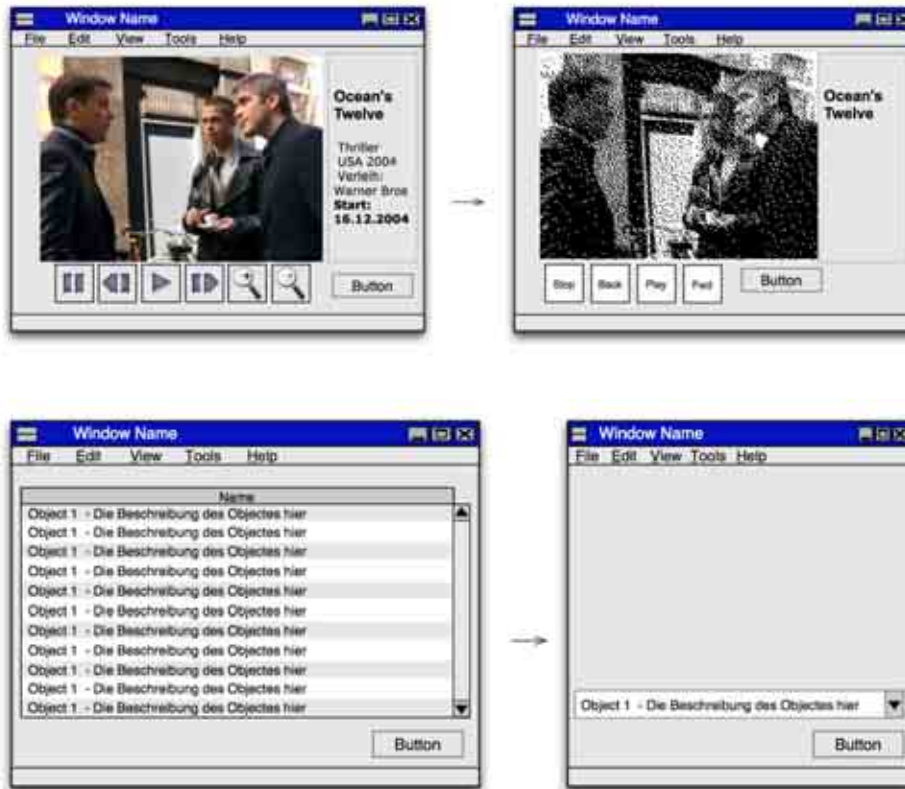


Abbildung 5.16: Vereinfachungsstrategien: Reduzierung des Inhaltes, Substitution.

5.3.5.2 Substitution

Lässt sich eine Information auf einem Gerät mit herkömmlichen Eingabeelementen nicht mehr darstellen, dann besteht die Möglichkeit, Standardelemente durch spezielle Elemente zu ersetzen und so die Information konzentriert, evtl. über andere Metaphern zu vermitteln. Ein Beispiel hierfür ist die Eingabe von Daten und Zeiten, welche häufig als platzaufwändige Kalender-Darstellungen benötigen. Auf begrenztem Raum können diese meist nicht mehr dargestellt werden und müssen durch angepasste Elemente ersetzt werden [VLF⁺01, BCCR04, NML04, BI03].

Die hier aufgeführten Transformationsstrategien dienen in erster Linie der Veranschaulichung der Transformationsannahme. Es sollte verdeutlicht werden, wie sich auch komplexe Anpassungen in einzelne Transformationsschritte zerlegen lassen. Die Analyse komplexer Anpassung in einzelne Transformationen ist insbesondere dann notwendig, wenn man heterogene Anpassungen vereinheitlichen möchte, um plattformübergreifende Konsistenz zu erreichen. Wie bereits argumentiert wurde, ist eine transparente und nachvollziehbare Anwendung von Transformationen notwendig, um es dem Benutzer zu ermöglichen Anwendungswissen zwischen Instanzen einer Anwendung einfach übertragen zu können.

Die hier geschaffene Übersicht ist in erster Linie an den graphischen Aspekten der Ordnung und Größe einer Oberfläche orientiert und versucht die hier vorstellbaren Mechanismen zu klassifizieren. Dieses Schema erhebt keinen Anspruch auf Vollständigkeit und eine Erweiterung um andere Aspekte scheint denkbar und nötig. Der heuristische Wert der Klassifikation steht hierbei deutlich im Vordergrund und wird im Rahmen der Konzeptvalidierung in Abschnitt 7 noch deutlicher werden.

5.4 Plattformübergreifende Konsistenz

Jedes Bild, das man zum erstenmal sieht, ist eine Art Herausforderung. Man muß sich darin zurechtfinden, muß die Struktur entdecken, die einem den Sinn des Werkes eröffnet.

— Rudolf Arnheim, *Die Macht der Mitte*, 1982

Ein zentrales Problem bei der Entwicklung von plattformübergreifenden Benutzerschnittstellen besteht darin, die Optimierung für eine Darstellung und die übergreifende Konsistenz gegeneinander abzuwägen. Diese Problematik wurde in den Anforderungen an den Entwicklungsprozess in Abschnitt 4.4.1.2 als *Optimierungsproblem* [FAPQA04] beschrieben. Insbesondere modellbasierte UIMS, welche auf die automatische Generierung einer Darstellung aus einer abstrakten Beschreibung aufbauen, stehen häufig vor der Problematik, dass die automatisch generierten Darstellungen nicht den Anforderungen der Designer entsprechen. Änderungen in der endgültigen Darstellung hingegen sind meist nicht vorgesehen und eine Unterstützung bei der Abwägung von plattformübergreifender Konsistenz und Design existiert bislang nicht.

In dem hier beschriebenen Konzept der plattformübergreifenden Entwicklung (s. Abschnitt 5.2) wird aus diesem Grund ein Konzept der geordneten Entwicklungspfade (s. Abschnitt 5.2.2) vorgeschlagen. Ein Pfad (d. h. die Darstellung für eine Plattform) ist dominant, die anderen werden als Ableitungen dieses Pfades betrachtet. Die Ableitungen können im Sinne von Transformationen (s. Abschnitt 5.2.3) dargestellt werden. Die Konsistenz zwischen den Plattformen kann nun als die Konsistenz der Ableitung betrachtet werden. Sie wird dem Designer als Anhaltspunkt für die gezielte Anpassung der generierten Designs angeboten.

In den folgenden Abschnitten wird zunächst die Konsistenz im Kontext der plattformübergreifenden Entwicklung diskutiert. Dann wird basierend auf diesen Überlegungen, ein Maß für die Beschreibung der plattformübergreifenden Konsistenz entwickelt und exemplarisch beschrieben.

5.4.1 Konsistenz plattformübergreifender Benutzerschnittstellen

Der Begriff der plattformübergreifenden Konsistenz findet insbesondere seit den Arbeiten von Denis und Karsenty [DK04] zur *Inter-Usability* plattformübergreifender Systeme verstärkt Beachtung. Das Thema der plattformübergreifenden Konsistenz wurde in den letzten Jahren in verschiedenen Foren in verstärktem Maße diskutiert und beispielsweise auf dem *Workshop on Multi-User and Ubiquitous User Interfaces 2004 (MU3I)* als einer der wesentlichen Herausforderungen der plattformübergreifenden Entwicklung von Benutzerschnittstellen erachtet [BKKS04].

Trotz einiger wegweisender Arbeiten, die hier kurz angesprochen werden sollen, ist eine allgemein anerkannte Definition des Begriffes bislang nicht erfolgt. Die Betrachtungsweise unterscheidet sich zwischen den Arbeitsgruppen oft noch erheblich, wobei in manchen Fällen die Konsistenz der Dialogstruktur [NM05], die Ähnlichkeit der Darstellung [GWW05] oder die Kontinuität [FTV04] im Vordergrund steht.

5.4.1.1 Konsistenz der Dialogstruktur

Myers und Nichols konzentrieren sich in ihrer Betrachtung der Konsistenz auf die Fragen, *können Dialoge überhaupt konsistent sein, wenn sich diese in Ihrer Funktionalität unterscheiden, welche Dimensionen der Konsistenz sind relevant, wie viel Übung mit einem System ist notwendig, damit Konsistenz einen Effekt hat* [Mye01, NM05].

Die Frage, ob Konsistenz bei verschiedenem Funktionsumfang möglich ist machen Nichols und Myers abhängig von der Art und Weise in welcher die Gruppierung der Element im Interface erfolgt. Sie unterscheiden hier zwischen geringer Ähnlichkeit (*sparse*), Verzweigungsähnlichkeit (*branch*) und deutlicher Ähnlichkeit (*significant*). Anwendungen mit geringer Ähnlichkeit werden versuchen, die Darstellung der Elemente beizubehalten. Anwendungen mit Verzweigungsähnlichkeit werden versuchen, die Struktur der Dialoge gleich zu halten und neue Inhalte in die alten Strukturen einzupassen. Deutliche Ähnlichkeit hingegen benötigt sowohl die strukturelle als auch die visuelle Ähnlichkeit.

Myers und Nichols formalisieren diesen Ansatz nicht weiter aus, so dass dieser Ansatz zwar konzeptionell interessant, praktisch jedoch nicht umsetzbar bleibt.

5.4.1.2 Ähnlichkeit der Darstellung

Gajos *et al.* [GWW05] betrachtet Konsistenz aus der Perspektive der *Ähnlichkeit*. Diese Sichtweise erklärt sich in erster Linie aus dem Ansatz, welchen diese Gruppe der *University of Washington* mit dem SUPPLE [GW04] System für automatisches Layout von User Interfaces verfolgt. Gajos versteht die Erzeugung und Darstellung von Dialogen als Optimierungsprozess. Das SUPPLE System erzeugt ein Layout in erster Linie als Optimierung verschiedener Darstellungsparameter. Hierbei treten selbstverständlich all jene Problematiken auf, welche in Abschnitt 2.4 für die automatische Generierung von Dialogen diskutiert wurden.

Insbesondere bei einer Optimierung muss stets zwischen verschiedenen Aspekten abgewogen werden, wobei die Anteile und Resultate der verschiedenen Einflussgrößen sich von Fall zu Fall stark unterscheiden können. Die Einbeziehung der Konsistenz in diesen Optimierungsprozess stellt aus diesem Grunde eine besondere Herausforderung dar. Gajos beschreibt Konsistenz über eine *Unähnlichkeitsfunktion*, welche als Summe aller Unterschiede über die Elemente einer Anwendung berechnet wird.

In einer ersten explorativen Studie untersuchten Gajos *et al.* [GWW05] verschiedene Dialoge und verglichen diese auf Ähnlichkeit, um so die für die Ähnlichkeitsbeurteilung relevanten Eigenschaften der Dialoge zu extrahieren. Als wichtige Kriterien identifizierten sie die *Sprache*, bzw. die Metapher, mit der der Wert des Elements vermittelt wird (*Toggle*, Text, Position eines Schiebereglers, etc.), die *Sichtbarkeit des Wertebereichs* (voll, teilweise, nur den aktuellen Wert), die *Orientierung der Datenpräsentation* (horizontal, vertikal, zirkulär), die *Skalenart* (diskret, kontinuierlich), die *Variabilität der Werte* (Listeninhalte können ausgetauscht werden, Radio Buttons sind statisch), die *Eingabemethode* (Auswahl, Drag, Tastatur), und die *Ausrichtung* des Elements (horizontal, vertikal).

All diese Werte, so nehmen Gajos *et al.* an, können ausschlaggebend für die Ähnlichkeitswahrnehmung sein. Designerurteile sollen diese Hypothese stützen. Rahmen eines automatisierten Layout-Prozesses kann eine Konsistenz-, bzw. hier Ähnlichkeitsbewertung zum einen als Teil des Optimierungsprozesses eingesetzt werden. Eines der Maße, die also optimiert werden, ist die Ähnlichkeit zwischen Plattformen. Wie meist bei solchen Optimierungsprozessen bleibt es jedoch eine Frage der Gewichtung, wie stark die Ähnlichkeit in den Layoutprozess eingeht. Die Gefahr ist also, dass die Ähnlichkeit nur bei bestimmten Konstellationen ausreichend Berücksichtigung findet und somit unzuverlässig wirkt. Eine alternative Einbeziehung des Konsistenzmaßes kann also auch hier in der *post-hoc* Bewertung der generierten Darstellung liegen, welches dann vom Designer nachträglich korrigiert werden kann.

5.4.1.3 Kontinuität der Interaktion

Während die Interaktion mit heutigen WIMP System meist auf der Auslösung von Ereignissen und damit verbundenen Funktionsaufrufe besteht, haben vor allem natürliche Interaktionen die Eigenschaft der Kontinuität. Werden verschiedene Ein- und Ausgaben berücksichtigt und kombiniert sollte hierbei die Interaktion keine harten Übergänge beinhalten um diese nicht zu stören.

Massing und Faconti [MF02] präsentieren ein konzeptionelles Rahmenmodell wie diese Kontinuität unterstützt werden kann. Florins *et al.* [FTV04] betrachten, aufbauend auf den Arbeiten von Denis und Karsenty [DK04], die Kontinuität der Interaktion, als Kontinuität der verschiedenen Ebenen des Dialoges, als ausschlaggebend. Florins *et al.* generalisieren dieses Problem auch auf andere Anwendungen mit wechselnden Kontexten, wie z. B. *Mixed Reality* Anwendungen.

5.4.1.4 Konsistenz als Aspekt der *Inter-Usability*

Denis und Karsenty [DK04] stellen in ihrem Beitrag ein erstes umfassendes Konzept der Benutzbarkeit multiple Benutzerschnittstellen (*Multiple User Interfaces*) auf. Plattformübergreifende Benutzerschnittstellen können als Teilmenge dieser Fragestellung betrachtet werden.

Denis und Karsenty [DK04] stellen fest, dass der Übergang zwischen verschiedenen Plattformen, und die Implikationen für den Benutzer dabei noch kaum erforscht sind. In plattformübergreifenden Umgebungen, genügt es nicht, die Benutzbarkeit der einzelnen Plattformen zu verbessern. Der Übergang zwischen den Plattformen sollte für den Nutzer so einfach und wenig aufwändig wie möglich sein. Denis und Karsenty führen das Konzept der *Inter-Usability* ein, mit welchem der Aufwand beschrieben wird, den der Nutzer beim Wechsel zwischen zwei Plattformen hat und in welchem Umfang er bereits vorhandenes Wissen übertragen kann. Seffah und Javahery definieren hierfür den Begriff *Horizontale Usability* [SJ04b].

Wird der Benutzer das erste Mal mit einem Problem konfrontiert, wendet er Problemlösungsmechanismen an, um dieses zu lösen. Dieser Vorgang ist aufwändig und kann mit der Notwendigkeit zur Exploration verbunden sein. Hat der Benutzer nun aber eine Lösung erarbeitet, integriert er sie in seine Sicht des Systems (s. Abschnitt 3.2 und 3.3). Wird er das nächste Mal mit einem ähnlichen Problem konfrontiert werden, wird er versuchen die erarbeiteten Lösungsstrategien erneut anzuwenden. Diese müssen zum Teil zunächst an die gegebenen Umstände angepasst werden. Hierzu muss der Kontext für den Benutzer transparent sein, er muss beispielsweise feststellen können, ob notwendige Voraussetzungen erfüllt sind.

Im Falle des Wechsels zwischen Plattformen bedeutet dies, dass der Benutzer sein Wissen an die neue Plattform und deren Spezifika anpassen muss. Dies kann erleichtert werden, wenn die Systemeigenschaften und Zustände dem Benutzer *transparent* sind. Die Erfahrung des Benutzers ist hierbei ein wichtiger Faktor, je weniger Erfahrung der Benutzer auf einer bestimmten Plattform hat, desto stärker wird er von Hinweisen in der Benutzungsoberfläche abhängen. Je fließender der Übergang zwischen Plattformen, desto einfacher fällt es dem Benutzer, vorhandenes Wissen zu übertragen. Denis und Karsenty bezeichnen dies als die Kontinuität des Wissens und die Kontinuität der Daten und Aufgaben.

Kontinuität des Wissens basierend auf dem Wissen des Benutzers, welches sich dieser bei der Nutzung dieser oder anderer Plattformen angeeignet hat.

Visuelle Erscheinung und Benennung Oberflächenelemente und deren Organisation können sich über verschiedenen Plattformen unterscheiden in *Räumlicher Anordnung* und *Form*. Nicht alle graphischen Unterschiede sind in demselben Maße kritisch, so wirken sich Unterschiede in Skalierung, Farbe und Orientierung meist nicht allzu negativ aus. Terminologie, die Benennung von Elementen in der Oberfläche, bzw. die Nutzung von Oberflächenelementen sollte konsistent über die verschiedenen Plattformen erfolgen.

Trennung der Daten und Funktionen Daten und Funktionen sind nicht auf jeder Plattform in demselben Maße verfügbar. Der Benutzer wird also beim Wechsel zwischen Plattformen auf Funktionen stoßen, die nach dem Wechsel neu hinzugekommen sind oder nun nicht mehr zur Verfügung stehen.

Redundante Geräte alle Geräte haben denselben Funktionsumfang. Dennoch kann der Benutzer aufgrund der Unterschiede im Zugriff Probleme bekommen.

Exklusive Geräte jedes der Geräte hat einen völlig unterschiedlichen Funktionsumfang. Kommt selten vor (evtl. versch. Views auf Daten).

Komplementäre Geräte der Funktionsumfang der Geräte überschneidet sich zum Teil. Funktionen, die auf einem Gerät vorhanden sind stehen auf dem anderen Gerät nicht zur Verfügung.

Prozedurale Konsistenz neben der Übereinstimmung der Funktionalität sollte auch die Interaktion an sich konsistent sein. Die prozedurale Konsistenz muss nicht dazu führen, dass eine Aktion auf allen Plattformen auf dieselbe Art durchgeführt werden muss, da dies möglicherweise nicht optimal für eine Plattform ist. Vielmehr sollte aber die Variation konsistent sein. Unterscheidet sich die Interaktion in einem Fall auf eine bestimmte

Weise, sollte sie in einem entsprechenden Falle ähnlich angepasst sein. Zusätzlich kann Hilfe und Transparenz bei der Umstellung helfen.

Kontinuität der Daten und Aufgaben Status und Daten vergangener Interaktionen, welche jeder Plattform zur Verfügung stehen sollten, um den Eindruck eines gemeinsamen Datenspeichers zu erwecken.

Weiterführung der Datenverfügbarkeit wird die Interaktion auf einem Gerät abgebrochen und auf einem anderen Gerät fortgeführt, soll der Zustand der Daten möglichst ohne Brüche weitergeführt werden.

Weiterführung des Handlungskontextes wird die Interaktion von einem Gerät auf ein anderes fortgesetzt, ist es möglicherweise auch wichtig, dass die zuletzt durchgeführte Handlung wieder verfügbar ist, ohne dass alle Vorbedingungen erneut erfüllt werden müssen.

Insbesondere die Konsistenz der *visuellen Erscheinung* ist im Rahmen dieser Arbeit von großer Bedeutung. Aus diesem Grunde soll hier auf die Implikationen der räumlichen Organisation und der Form nach Denis und Karsenty nochmals detailliert eingegangen werden.

Unterschiede in der *räumlichen Organisation* bedeuten, dass ein Objekt in verschiedenen plattform-spezifischen Versionen einer Anwendung oder eines Dienstes nicht an derselben Stelle zu finden sind. Dies bedeutet für den Benutzer, dass er das Objekt suchen muss. Damit verbunden ist eine Erhöhung des kognitiven Aufwands bei der Benutzung und damit stärkere Ermüdung, Anfälligkeit gegen Ablenkung, usw. Im schlimmsten Falle kann die Unauffindbarkeit einer vorhandenen Funktion fälschlich in dem Sinne interpretiert werden, dass diese nicht vorhanden sei. Dies kann sich fatal auf die Benutzbarkeit einer Plattform auswirken, insbesondere bei der Frage für den Benutzer, wann die Suche nach einer Funktion abgebrochen werden soll.

Unterschiede in der *Form* eines Elementes der Oberfläche auf verschiedenen Plattformen kann die Wiedererkennbarkeit gefährden. Der Benutzer kann so Probleme bekommen, eine bekannt Funktion einem Element zuzuordnen und diese wiederum als nicht vorhanden interpretieren. Ein hoher Zeitaufwand durch Suche und Exploration ist die Folge. Denis und Karsenty ziehen daraus die Schlussfolgerung, dass diese Dimensionen möglichst konstant gehalten werden müssen zwischen Plattformen.

Beim Wechsel zwischen Plattformen soll der Nutzer also sein existierendes Wissen möglich umfassend weiterverwenden können. Aufgrund systemspezifischer Besonderheiten, können identische Oberflächen aber nicht angeboten werden, es gibt Einschränkungen bei der Darstellung, beim Funktionsumfang und bei der Interaktionsweise. Plattformübergreifendes Design muss also Inkonsistenzen akzeptieren und mit ihnen umgehen. Eine Möglichkeit ist die der *Transparenz*. Dem Benutzer muss die Systematik und evtl. auch der Grund der Inkonsistenz klar sein.

Transparenz definieren Denis und Karsenty im Sinne von Maass [Maa83] „a transparent system makes it easy for users to build up an internal model of the functions the system can perform for them.“

Die *Transparenz* kann also beschrieben werden als eine Eigenschaft des Mensch-Maschine Dialogs, welche es dem Nutzer ermöglicht, sich eine akkurate Repräsentation des Systems zu erstellen und somit effizient mit dem System zu interagieren. Transparenz kann dazu nützlich sein, dem Nutzer Unterschiede zwischen Plattformen zu erklären, d. h. *transparent zu machen*. Dies kann über Hilfen, aber auch über Hinweise in der Oberfläche geschehen. Die Transparenz hängt von der Expertise des Nutzers ab und sollte aus diesem Grunde adaptierbar sein.

Da die Transparenz auch davon abhängt, welche Vorkenntnisse der Benutzer von dem System, bzw. von der Plattform hat, ist Adaptierbarkeit an den Benutzer und dessen Vorkenntnisse notwendig. Denis und Karsenty sprechen also von drei zentralen Design-Prinzipien zur Entwicklung von plattformübergreifenden Anwendungen: *Plattformübergreifende Konsistenz*, *Transparenz* und *Adaptierbarkeit* des Dialogs.

Die *Konsistenz* zwischen den Plattformen soll dazu dienen, dass der Benutzer beim Wechsel einen hohen Wiedererkennungsgrad hat und dadurch auf bereits vorhandene Erfahrung von anderen Plattformen weiter nutzen kann. Denis und Karsenty identifizieren vier Ebenen der plattformübergreifen-

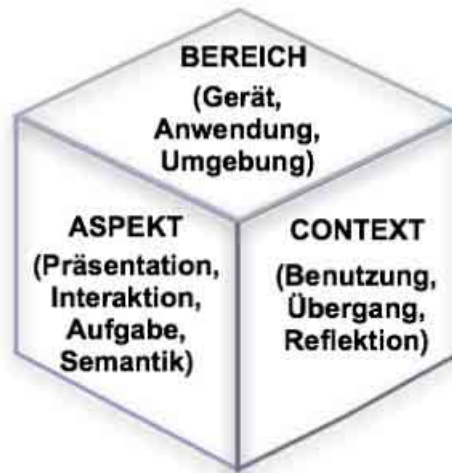


Abbildung 5.17: Ein Referenzmodell zur Klassifikation der Dimensionen der Konsistenz multipler Anwendungen. Der Bereich beschreibt, welche Instanzen eines Aspektes gegenüber gestellt werden, der Aspekt beschreibt das Maß welches verglichen wird, und der Kontext bestimmt welche Nutzungsphase betrachtet wird.

den Konsistenz, welche deutlich an das konzeptuelle Rahmenmodell von Kellog [Kel87] angelehnt sind.

Semantische Konsistenz Dienste und Anwendungen sollten auf verschiedenen Plattformen gleich sein, d.h. die Aufteilung von Daten und Funktionen sollte gleich sein und sich weitestgehend überlappen (s.o. Redundante Geräte).

Syntaktische Konsistenz über die Plattformen hinweg sollten gleiche Ziele über die selben Interaktionen durchgeführt werden. Sollte durch eine spezielle Anpassung, die Interaktion auf einer Plattform wesentlich vereinfacht werden, kann diese natürlich eingeführt werden auch wenn sie inkonsistent ist. Dann sollte dennoch auch der konsistente (erwartungskonforme) Weg möglich sein, bzw. die Anpassung für mehrere Aktionen möglich sein und konsistent angewendet werden.

Lexikalische Konsistenz Konsistente Benennung und textuelle Inhalte.

Perzeptuelle Konsistenz Erscheinung und visuelle Struktur der Anwendungsoberfläche.

Die *Adaptierbarkeit* von Dialogen soll ermöglichen, dass beim Wechsel zwischen Geräten bisherige Erfahrungen des Nutzers mit dem System (d. h. der Anwendung) oder Geräten berücksichtigt werden. Kennt er sich mit bestimmten Geräten bereits aus, kann man davon ausgehen, dass er bestimmte Konzepte bereits geübt hat und bestimmte Einschränkungen ihm bereits geläufig sind. Hierzu schlagen Denis und Karsenty ein zentrales Benutzermodell vor, welches die Anpassung auf die Geräte parametrisieren kann.

5.4.2 Referenzmodell der Konsistenz multipler Benutzerschnittstellen

Wie sich gezeigt hat, unterscheiden sich die verschiedenen Ansätze die Konsistenz multipler Anwendungen zu beschreiben in erheblichem Maße. Dennoch liegt allen Ansätzen offensichtlich dasselbe Konzept zugrunde. Eine Vereinheitlichung und Einordnung erscheint daher wünschenswert und auch zulässig.

Richter *et al.* [RNGS06a, RNGS06b] beschreiben ein allgemeines Referenzmodell zur Einordnung der verschiedenen Aspekte multipler Konsistenz, welches versucht die verschiedenen Perspektiven in ein gemeinsames Bezugssystem zu setzen. Das Referenzmodell definiert drei Dimensionen, in die sich die oben beschriebenen Aspekte einordnen lassen. Abbildung 5.17 gibt eine Übersicht über die Dimensionen der Konsistenz.

5.4.2.1 Bereich

Die bekannteste, wenn nicht *die* prototypische Form der Konsistenz, betrachtet die Gestaltung einer Anwendung auf einem Endgerät. Diese Form wurde als *Anwendungskonsistenz (intra-application)* bezeichnet. Anwendungskonsistenz ist eine Voraussetzung für jede Form der plattformübergreifenden Konsistenz. Etwas weiter greift die *Plattformkonsistenz (intra-device)*, die über Style Guides, Anforderungen und Gestaltungskonventionen für eine konsistente Gestaltung innerhalb einer Geräteplattform sorgt. Diese Form wird auch vertikale Konsistenz genannt. *Plattformübergreifende (cross-device)* Konsistenz schließlich beschreibt das Verhältnis Zwischen Anwendungen auf verschiedenen Endgeräten. Diese Beziehung als eigentlichem Kernthema dieser Arbeit, kann freilich nicht isoliert von den vorangegangenen Perspektiven betrachtet werden.

Die verschiedenen Perspektiven oder Vergleichspaare, die in den verschiedenen Perspektiven gegenübergestellt werden, spannen einen *Bereich* auf. Ein Bereich in dem eine Konsistenzbeziehung gilt. Aus diesem Grund wird als erste Dimension des Referenzmodells der Bereich festgeschrieben. Als Bereiche kommen auch soziale Akteure, wie der Benutzer oder die Gesellschaft in Frage. Der *Bereich* definiert sich wie folgt:

Der Bereich der Konsistenz beschreibt den Raum, in welchem die Konsistenzbeziehung betrachtet wird. Er wird meist durch die Betrachtungspaare, wie Plattformen, Anwendungen oder Elemente einer Anwendung beschrieben.

5.4.2.2 Aspekt

Konsistenz betrifft alle Ebenen der Mensch-Maschine Kommunikation. Während in der Regel die Darstellung und die Terminologie im Vordergrund der Betrachtung stehen, so können auch Aspekte höherer Ebenen, wie die Aufgabenstruktur oder Semantik betroffen sein. Aufgabenkonsistenz zum Beispiel beschreibt die Übereinstimmung der, in der Anwendung implementierten Aufgabe mit der Aufgabe, die der Benutzer in seinem Kopf hat.

Der Aspekt der Konsistenz beschreibt die Ebene der Mensch-Maschine-Kommunikation, die hinsichtlich der Übereinstimmung, bzw. Regelmäßigkeit betrachtet wird. Er wird meist durch Konzepte wie Darstellung, Terminologie, Aufgabenstruktur oder Bedeutung beschrieben.

5.4.2.3 Kontext

In welcher Nutzungsphase eine Konsistenzbeziehung bewertet wird, spielt ebenfalls eine wesentliche Rolle. Betrachtet man Konsistenz während der Nutzung, während des Wechsels zwischen Geräten, oder bei der Vermittlung von und Kommunikation über Geräte, können verschiedenen Aspekte und Bereiche völlig verschieden gewichtet werden. So scheint beispielsweise die Kontinuität stärker auf den Wechsel zwischen Geräten fokussiert, und hier Darstellung wesentlich wichtiger als bei der Anwendungskonsistenz, die stärker die Erlernung und dauerhafte Nutzung einer Anwendung unterstützt, wobei hier Aufgabenstruktur als tieferer Aspekt dauerhafter zum tragen kommen kann.

Der Kontext der Konsistenz definiert sich die Nutzungsphase für welche eine Betrachtung ange stellt wird. Schwerpunkte sind hierbei die dauerhafte Nutzung, der Wechsel und Erlernung, bzw. die Vermittlung und Kommunikation über eine Schnittstelle.

Vergleicht man dieses Referenzmodell mit den Dimensionen Kelloggs [Kel87], stellt man fest, dass diese Modell sich in erster Linie in der Dimension *Aspekt* wiederfindet. In Erweiterung des Modells von Kellogg sind die Dimensionen *Bereich* und *Kontext* notwendig geworden, um plattformübergreifende Konsistenz, bzw. verschiedene Nutzungsphasen beschreiben zu können. Wie gezeigt wurde, ermöglicht es dieses Referenzmodell, andere Perspektiven und Forschungsansätze einzuordnen und miteinander in Beziehung zu setzen.

5.4.3 Ein transformationales Konsistenzmaß

In Abschnitt 5.2.3 wurde das Transformationsparadigma beschrieben. Dieses Konzept vereint technische Vorgehensweisen der Software-Entwicklung mit Anforderungen die aus der Psychologie der Wahrnehmung und Informationsverarbeitung bei plattformübergreifender Darstellungen entstehen. Die Betonung des fachübergreifenden Ansatzes steht hierbei im Sinne einer benutzerzentrierten Entwicklungsstrategie im Vordergrund.

Auf Basis der Kriterien der visuellen Erscheinung von Denis und Karsenty [DK04], welche im vorigen Abschnitt ausführlich beschrieben wurden, wurden in Abschnitt 5.3 Transformationsstrategien zur Klassifikation von Abbildungen zwischen plattformspezifischen Darstellungen vorgestellt. Aufbauend auf diesen Konzepten und auf den Erkenntnisse aus der explorativen Voruntersuchung (s. Abschnitt 2.2) wird ein *transformationaler Ansatz zur Konsistenzmessung* vorgeschlagen [Ric06].

5.4.3.1 Definition der transformationalen Konsistenz

Denis und Karsenty stellen im Sinne einer klassischen Definition von Konsistenz fest, dass die Beibehaltung von Position und Form von Elementen für eine plattformübergreifend konsistente visuelle Gestaltung notwendig ist. Plattformspezifische Anpassungen der Darstellung müssen sich dem Benutzer *transparent* darstellen. So soll die daraus entstehende Inkonsistenz (d. h. unähnliche Gestaltung) ausgeglichen werden. Wesentlich für diesen Ansatz ist hierbei erneut die Reduzierung der Konsistenz auf die Ähnlichkeit verschiedener Aspekte. Die Forderung nach Transparenz kann aus diesem Grunde auch als Forderung nach der Erweiterung der klassischen Vorstellung von Konsistenz um ein Konzept, welches Abweichungen von einer Identitätsabbildung zu fassen ermöglicht, interpretieren. Dieser einschränkende Ansatz soll durch den transformationalen Ansatz, welcher in dieser Arbeit entwickelt wurde erweitert und ersetzt.

Die Definition der Konsistenz im Sinne einer konsistenten Transformation verschiebt der Fokus von der Identität der Darstellung hin zu der *Konstanthaltung der Abbildung* zwischen Plattformen. Nicht die Konstanthaltung der *Darstellung*, welche in der Regel auf Grund von plattformspezifischen Anforderungen nicht möglich ist, sondern die Konstanthaltung der *Abbildung* kann als Maß der Konsistenz plattformübergreifender Darstellung dienen. Dieser Ansatz wird hier im Kontrast zu dem klassischen Konsistenzbegriff als *transformationaler Konsistenzansatz* bezeichnet.

Bei der Konzeption des Konsistenzmaßes wird an dieser Stelle von bestimmten Randbedingungen ausgegangen. Der Benutzer besitzt Vorkenntnisse in einer Anwendung, die er auf einer bestimmten Plattform regelmäßig nutzt. Diese Plattform wird als die *Originalplattform* bezeichnet. Die Plattform, auf die der Benutzer als nächstes wechselt ist hingegen die abgeleitete *Variantenplattform*. Das Konzept der plattformübergreifenden Entwicklung (s. Abschnitt 5.2) berücksichtigt dies mit dem Ansatz der geordneten Entwicklungspfade.

Definition

Bei einer Abbildung wird die Ausgangsdarstellung als **Original** und die Zieldarstellung als **Variante** bezeichnet. Das Kriterium welches betrachtet wird, wird als der **Aspekt** der Abbildung bezeichnet. Die Abbildung eines oder mehrerer Aspekte wird über einen **Transformationsvektor** beschrieben.

Weiterhin wird in erster Linie auf die graphisch-interaktiven Aspekte, d. h. auf die visuelle Darstellung einer Oberfläche, nicht auf die zugrunde liegenden Modelle eingegangen. Das Anwendungswissen, welches hier diskutiert wird, bezieht sich also in erster Linie auf die syntaktischen und lexikalischen Aspekte im Sinne von Foleys Ebenenmodell [FD82], bzw. der Ebene des *Concrete User Interface* und der finalen Präsentation des CAMELEON Rahmenmodells modellbasierter Entwicklung [TCC04].

Die Transformationshypothese (s. Abschnitt 5.2.3) geht davon aus, dass der Benutzer den Wechsel zwischen Geräten stets als eine Transformation zwischen den gerätespezifischen Darstellungen der

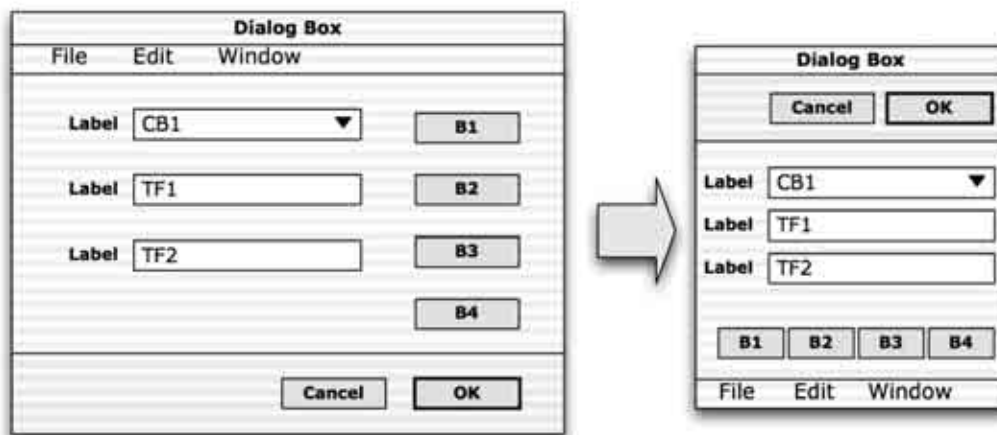


Abbildung 5.18: Beispiel für die Anpassung eines User Interface an eine neue Plattform.

Anwendung wahrnimmt (hierzu Experiment 7.2). Ist der Benutzer in der Lage, die Regelmäßigkeit der Abbildung zu erkennen, kann er diese nutzen, um sein mentales Modell der Anwendung zwischen den Plattformen anzupassen.

Hieraus leitet sich also im Gegensatz zu Denis und Karsenty [DK04] die Annahme ab, dass nicht unbedingt die *Beibehaltung* von Position, Form und Anzahl, sondern vielmehr die *konsistente Abbildung*, bzw. *konsistente Transformation* dieser Dimensionen zwischen den verschiedenen Darstellungen für eine konsistente plattformübergreifende Darstellung notwendig ist.

Das hier vorgestellte Konsistenzmaß beruht also nicht auf einem *absoluten*, sondern vielmehr auf einem *relativen* Vergleich der plattformspezifischen Darstellungen. Die *klassische Konsistenzvorstellung* ist damit zugleich ein Sonderfall der *transformationalen Konsistenz* für den Fall, dass die Abbildung zwischen den Plattformen eine Identitätsabbildung ist.

Definition

*Das transformationale Konsistenzmaß beschreibt die **Regelmäßigkeit** und nicht die Ähnlichkeit der Abbildung eines Originals in eine Variante.*

Ein einfaches Beispiel soll hier zur Verdeutlichung dienen. Betrachten wir die Abbildung zwischen einer Anwendung auf einem Desktop PC und einem Pocket PC-Mobilgerät mit den jeweiligen Versionen des Microsoft Windows Betriebssystems. Während die Menüleiste bei der Desktop-Version stets am oberen Rand des Fensters platziert ist, befindet sich diese auf dem Pocket PC stets am unteren Rand. Aufgrund der Bedienung des Pocket PC mit Stifteingabe ist diese Anpassung ergonomisch sinnvoll, da so bei der Auswahl eines Menüelements nicht stets der gesamte Bildschirm und somit auch das Menü selbst von der Hand verdeckt wird.

Nach dem klassischen Verständnis von Konsistenz ist diese Abbildung inkonsistent, denn die Anordnung des Elementes wurde nicht konstant gehalten. Im Sinne der transformationalen Konsistenzmessung ist dies jedoch eine konsistente Abbildung. Der Benutzer kann diese Transformation, eine *Translation*, schnell ableiten und anhand dieses Wissens sein Anwendungswissen anpassen: *Wenn das Menü auf dem Desktop oben war ist es auf dem Pocket PC unten. Suche ich eine Funktion im Menü, muss ich anstatt oben nun unten suchen.* In diesem Fall wurde die Transformation der Aspekt *Anordnung*, oder spezieller des Kriteriums *Position*, betrachtet. Diese Transformation ist Beispiel eindeutig. Die Vorhersage der Position des Menüs auf der Pocket PC-Plattform ist somit für den Betrachter mit großer Sicherheit möglich.

Ein weiteres Beispiel veranschaulicht das Prinzip der transformationalen Konsistenz anhand des Aspektes *Form*. Angenommen zur Realisierung einer Auswahl wird auf dem Desktop PC stets ein

Listenelement verwendet. Ein solches Listenelement ist in der Regel recht platzaufwändig. Gleichzeitig ermöglicht es, eine große Anzahl von Auswahlalternativen anzuzeigen. Da der Pocket PC nur geringen Bildschirmplatz zur Verfügung stellt, wird hier häufig eine Liste durch ein kompakteres Element, eine Combobox ersetzt. Um Platz zu sparen ist diese Umsetzung also häufig notwendig. Auch diese Umsetzung resultiert im klassischen Sinn in einer inkonsistenten Abbildung. Die Form wird nicht konstant gehalten. Im transformationalen Sinn kann man dennoch von einer konsistenten Abbildung sprechen, wenn *alle* Listen auf dem Desktop PC in Comboboxes auf dem Pocket PC übersetzt werden.

Die transformationale Konsistenz ist umso höher, je größer die Vorhersagbarkeit der Form der Abbildung anhand der Form des Originals ist.

Das transformationale Konsistenzmaß erlaubt es, Anpassungen der Gestaltung, welche aufgrund plattformspezifischer Anforderungen notwendig sind (z.B. ergonomische Besonderheiten bei der Bedienung mit einem Stift) zu berücksichtigen ohne eine Darstellung als inkonsistent zu verwerfen. Das transformationale Konsistenzmaß kann auch als Maß der Vorhersagbarkeit der Abbildung beschrieben werden.

Die dieser Annahme zugrunde liegende Rationale ist die Folgende: Eine hohe Vorhersagbarkeit der Abbildung motiviert den Benutzer, vorhandenes Wissen für den Transfer auf die neue Plattform zu transformieren, anstatt neues Wissen aufzubauen.

Nach Johnson-Laird [JL83] (s. Abschnitt 3.2) wird neues Wissen in das existierende mentale Modell eingebunden indem dieses angepasst wird. Sind neue Erkenntnisse ausreichend *nahe* an dem vorhandenen Wissen, dann werden die neuen Inhalte in das existierende Modell integriert. Mit Hinblick auf die graphische Darstellung einer Oberfläche kann dies bedeuten, dass eine einfache Abbildung des Neuen in das Alte möglich ist.

Wird die Transformation einer Dimension über alle Dialoge einer Anwendung konsistent angewandt, dann ist diese Abbildung einfacher zu erkennen und zu generalisieren, als wenn sich diese Abbildung für jeden einzelnen Dialog unterscheidet. Im ersten Fall kann die Darstellung der Variante anhand derselben Transformation mental antizipiert und damit auch mit dem vorhandenen Wissen in Verbindung gebracht werden. Im zweiten Fall müssen beliebig viele Sonderfälle berücksichtigt werden, so dass eine Abbildung auf vorhandenes Wissen u. U. aufwändiger ist, als der Aufbau eines neuen Modells.

Betrachtet man also Konsistenz als Maß der Vorhersagbarkeit der Darstellung der Variante anhand der gegebenen Darstellung des Originals, kann plattformübergreifende Konsistenz als statistisches Maß beschrieben werden. Die bedingte Wahrscheinlichkeit einer Variante gegeben der Darstellung des Originals. Wahrscheinlichkeit wird für Elemente einer *Gruppe* auf der der Originalplattform über alle Dialoge einer Anwendung bezüglich derer korrespondierender Elemente auf der Variantenplattform definiert. Die Gruppe von Elementen definiert sich über ihre Einheitlichkeit bezüglich eines gegebenen Aspektes, z.B. deren Position, Größe, Form, oder auch einer Kombination dieser Maße. Eine Gruppe beinhaltet also alle Elemente über alle Dialoge einer Anwendung, welche sich in einem Maß gleichen.

Definition

Die transformationale Konsistenz ist ein Maß der Vorhersagbarkeit einer Variante anhand eines Originals.

Das Entwicklungskonzept (s. Abschnitt 5.2) sieht vor, dass jedes Element auf der Originalplattform ein korrespondierendes Element auf der Variantenplattform besitzt oder ausgelassen wird. Dies bedeutet, dass jedes Element der Gruppe bzgl. des gegebenen Maßes auf das korrespondierende Element auf der Variantenplattform abgebildet werden kann. Elemente der Gruppe mit dem Positionsvektor (x, y) werden also auf eine Gruppe von n korrespondierenden Elementen mit den Positionsvektoren $[(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)]$ abgebildet. Diese Abbildung kann als eine Menge von Translationsvektoren oder anderer Arten von Transformationsvektoren dargestellt werden [Ric06].

Die Konsistenz der Abbildung kann nun auf zwei mögliche Arten bestimmt werden: *varianzanalytisch* oder *stochastisch*.

5.4.3.2 Varianzanalytische Bestimmung der Konsistenz

Eine *varianzanalytische* Bestimmung der Konsistenz (C) basiert auf der Berechnung eines Streuungsmaßes für die Menge der einzelnen Transformationsvektoren. Ein mögliches Streuungsmaß wird über die Summe der quadratischen Abweichungen vom Mittelwert bestimmt, die Varianz (σ^2) berechnet sich aus der Summe der Abweichungsquadrate geteilt durch die Anzahl der Meßpunkte.

$$\begin{aligned} C &\approx \sigma^2 \\ &\approx \frac{\sum_{n=1}^n (x_i - \bar{x})^2}{n} \end{aligned} \quad (5.1)$$

Die Berechnung des Streuungsmaßes berücksichtigt die Beträge der Unterschiede in der Abbildung und ist somit inhaltsensitiv. Auf der anderen Seite kann eine einzige vom Betrag große Abweichung schnell zu einem niedrigen Konsistenzmaß führen und dieses u. U. überbewerten.

Problematisch ist ein varianzanalytisches Verfahren auch bei Abbildungen, welche nicht intervallskaliert, sondern ordinal- oder nominalskaliert sind. Auszählmethoden zur Annäherung der Varianz bei solchen Daten haben formal bereits den Charakter einer Wahrscheinlichkeitsbestimmung. Aus diesem Grund muss eine weitere Methode berücksichtigt werden, welche allein auf der Wahrscheinlichkeit einer Darstellung basiert und unabhängig vom verwendeten Skalenniveau ist.

Definition

Die varianzanalytische Berechnung des transformationalen Konsistenzmaßes erfolgt aus der Streuung der Transformationsvektoren.

5.4.3.3 Stochastische Bestimmung der Konsistenz

Die *stochastische Bestimmung* der Konsistenz berechnet sich als bedingte Wahrscheinlichkeit der häufigsten Abbildungsvariante gegenüber den allen anderen Varianten. Inhaltlich entspricht dies der Frage: *Wie wahrscheinlich ist die Darstellung V , wenn das Original O ist?* Die minimale Konsistenz ist bei dieser Methode abhängig von der Anzahl der Dialoge, da sie im Fall der Gleichverteilung über alle Dialoge vorliegt. Die Wahrscheinlichkeit ist dann $1/n$, wobei n die Anzahl der Dialoge ist.

$$\begin{aligned} C &\approx p(V|O) \\ &\approx \frac{p(V \cap O)}{p(O)} \end{aligned} \quad (5.2)$$

Wie bereits erwähnt, hat die stochastische Methode den Vorteil, dass sie unabhängig vom Skalenniveau der verwendeten Abbildungsmaße ist. Dadurch ist jedoch auch eine quantitative Bewertung des Zusammenhanges unmöglich. Ebenfalls ist die Verteilung der Wahrscheinlichkeiten über die verschiedenen Alternativen nicht berücksichtigt und somit eine kumulative Aussage nicht möglich.

Definition

Die stochastische Berechnung des transformationalen Konsistenzmaßes erfolgt aus der bedingten Wahrscheinlichkeit eines bestimmten Transformationsvektors.

5.4.3.4 Kognitive Last der Transformation

In weiterer Anwendung der Transformationshypothese kann nun die Annahme berücksichtigt werden, dass jede Transformation einen gewissen kognitiven Aufwand erfordert, also über ein spezifisches kognitives *Gewicht* verfügt. So mag beispielsweise eine Umsetzung einer Liste in eine Combox mag kognitiv aufwändiger sein als eine Translation um wenige Millimeter.

Somit ist nicht nur die konsistente Transformation, sondern auch der summierte kognitive Aufwand, welcher für diese Transformation notwendig ist, ausschlaggebend für die Konsistenz. Eine im klassischen Sinne konsistente Abbildung unter Beibehaltung von Form und Anordnung hat offensichtlich einen geringen kognitiven Aufwand, da hier die angewandte Transformation eine Identitätsabbildung darstellt. Die Summe der kognitiven Gewichte der einzelnen Transformationen ist somit (hypothetisch zumindest) gleich Null.

Werden bei der Abbildung mehrere Transformationen angewandt, ist das resultierende kognitive Gewicht gleich der Summe der kognitiven Gewichte der einzelnen Transformationen.

Erfolgt eine Abbildung konsistent kann eine Automatisierung, d. h. eine Anwendung von Regeln, genutzt werden um den kognitiven Aufwand zu reduzieren. Bei der Etablierung solcher Regeln ist ein initialer Lernaufwand notwendig, der der Ersparnis durch eine konsistente Abbildung zunächst entgegenwirkt.

Definition

Die kognitive Last einer Transformation stellt den Aufwand dar, eine mentale Abbildung einer Transformation zu erstellen.

5.4.4 Berechnung der transformationalen Konsistenz

Wie oben definiert wurde, ist die *transformationale Konsistenz* dadurch bestimmt, wie konsistent eine Transformation auf eine Gruppe von Elementen mit einer bestimmten Ausprägung eines Aspektes angewandt wird. Da die Konsistenz, stets die Beziehung zweier Systeme bezüglich eines Aspektes darstellt, beschreibt die transformationale Konsistenz also die Beziehung zweier Darstellungen bezüglich dieses gewählten Aspektes. Ein Aspekt kann hierbei auch eine Kombination aus mehreren einzelnen Aspekten (Position, Farbe, Orientierung, Beschriftung, etc.) sein.

Die Aspekte eines Elements werden durch Werte abgebildet. Diese Werte unterscheiden sich darin, welche Interpretation eine bestimmte Ausprägung dieses Wertes zulässt, d. h. welche mathematischen Operationen oder Vergleichsaussagen anhand dieser Werte möglich sind. Im meßtheoretischen Sinn spricht man hierbei von dem Skalenniveau eines Wertes [Bor89]. Anhand dieser Skalenniveaus lassen sich drei Typen von Merkmalen unterscheiden.

Kategorialskalierte Merkmale: \mathbb{K} Merkmale, die durch Werte beschrieben werden, die lediglich logische Gleichheits- bzw. Ungleichheitsaussagen zulassen. Diese werden auch als nominalskaliert bezeichnet. Diese Werte lassen keine mathematischen Operationen zu. Beispiele sind Kategorien wie männlich vs. weiblich, Raucher vs. Nichtraucher, etc.

Ordinalskalierte Merkmale: \mathbb{O} Merkmale, die durch Werte beschrieben werden, die eine monotone hierarchische Beziehung implizieren. Vergleichsurteile wie *kleiner als*, *größer als* und *gleich* sind möglich. Mathematische Operationen sind auch hier nicht möglich. Beispiele sind Windstärken oder Schulnoten.

Intervallskalierte Merkmale: \mathbb{I} Merkmale, deren Werte mathematische Operationen zulassen setzen die Gleichheit von Differenzen voraus. Die Differenz zwischen den Werten, die durch Zahlen abgebildet sind repräsentieren die Differenzen der wahren Merkmale (z.B. Temperatur, Zeit). Ein Spezialfall der Intervallskala ist die Verhältnisskala, bei der der Nullwert festgelegt ist (z.B. Länge, Gewicht).

Da in Abschnitt 5.4.3.2 bereits diskutiert wurde, dass der varianzanalytische Ansatz der Konsistenzmessung nur für intervallskalierte Daten zulässig ist, müssen diese Merkmalstypen im Folgenden gesondert behandelt werden.

Aspekte können alle Merkmale eines Elementes in einer Benutzungsoberfläche sein. In Abschnitt 4.4.1 wurden verschiedene Metriken zur Beschreibung von graphischen Oberflächen, wie z.B. Entfernung, Komplexität, Dichte, Gruppierung, Ausrichtung, etc., gesammelt. Diese Maße beziehen sich zum Teil auf einzelne Elemente, zum Teil auf Elementgruppen oder auch den gesamten Dialog. Die Analyse der Konsistenz erfolgt auf Ebene der einzelnen Elemente oder der Dialoge. Elementgruppen werden behandelt, indem die Werte der Gruppe auf die Gruppenmitglieder übertragen werden und die Analyse dann auf Ebene der Elemente stattfindet.

Ein Merkmal π wird also definiert als eine Menge von *Einzelmerkmalen*.

$$\pi \rightarrow (p_1, p_2, \dots, p_n) \quad (p \in [\mathbb{K}, \mathbb{O}, \mathbb{I}]) \quad (5.3)$$

Die Merkmale werden den Bestandteilen der Anwendungsoberfläche zugeordnet. Als oberste ordnende Einheit wird der *Dialog* δ festgelegt. Als Dialog werden meist Strukturen bezeichnet, die in der Aufgabenstruktur einer Anwendung einer Aufgabe entsprechen. Dies ist in graphischen Anwendungen in der Regel bei einem Fenster oder einem Unterdialog der Fall. Die Benutzerschnittstelle einer *Anwendung* (α) besteht in der Regel aus mehreren Dialogen.

$$\alpha \rightarrow (\delta_1, \delta_2, \dots, \delta_m) \quad (5.4)$$

Ein Dialog wiederum besteht aus einer Anzahl von *Elementen* ϵ , welche einzelne Ein- oder Ausgaben realisieren. Beispiele für Elemente in einer graphischen Benutzerschnittstelle sind Schaltflächen, Listen, Ausgabefelder, etc.

$$\delta \rightarrow (\epsilon_1, \epsilon_2, \dots, \epsilon_o) \quad (5.5)$$

Die Abbildung einer Anwendungsoberfläche von einer Plattform (Original = α^O) auf eine andere Plattform (Variante = α^V) kann also definiert werden als die Abbildung der Dialoge ($\delta_i^O \Rightarrow \delta_i^V$) aus denen die Anwendungsoberfläche bestehen.

$$\alpha^O \Rightarrow \alpha^V \quad (5.6)$$

$$\{\delta_1^O, \delta_2^O, \dots, \delta_m^O\} \Rightarrow \{\delta_1^V, \delta_2^V, \dots, \delta_m^V\} \quad (5.7)$$

Weiterhin kann die Abbildung der Dialoge auf die Abbildung der einzelnen Elemente ($\epsilon_{ij}^O \Rightarrow \epsilon_{ij}^V$) reduziert werden. Dem Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen (s. Abschnitt 5.2) folgend, kann die Gliederung der Elemente bzw. die Aufteilung in Dialoge in der Darstellung durch die Abbildung variieren (d. h. Elemente können in verschiedenen Dialogen oder Tabulatoren dargestellt werden).

$$\delta^O \Rightarrow \begin{cases} \emptyset \\ \delta^V \\ \{\delta_1^V, \delta_2^V, \dots, \delta_m^V\} \end{cases} \quad (5.8)$$

Die Abbildung der Elemente hingegen lässt keine $1 \rightarrow n$ Abbildung zu. Ein Element hat kein oder genau ein korrespondierendes Element. Der Gedanke ist der, dass ein Element in der semantischen Bedeutung bereits das kleinste möglich Element ist und somit nicht mehr in kleine Bestandteile aufgebrochen werden kann, bzw. soll.

$$\epsilon^O \Rightarrow \begin{cases} \emptyset \\ \epsilon^V \end{cases} \quad (5.9)$$

Die Transformation der Anwendungsoberflächen θ kann nun in Form der Transformationsstrategien, welche in Abschnitt 5.3 exemplarisch ausgeführt wurden, beschrieben werden.

Der Prozess der Konsistenzberechnung gliedert sich in vier Unterschritte:

1. Gruppierung
2. Berechnung der Transformation
3. Berechnung der Konsistenz
4. Auswertung und Aufbereitung

5.4.4.1 Schritt 1: Gruppierung

Wie bereits oben erwähnt, erfolgt die Berechnung der Konsistenz auf Ebene der Elemente oder der Dialoge. Hierzu werden in einem ersten Schritt die *Elemente der Analyse* (Dialoge oder Elemente) bezüglich des *Merkmals der Analyse* (z.B. Position, Beschriftung, Größe, Farbe) zu Gruppen γ gleicher Merkmale zusammengefasst. Bei der Gruppierung auf Elementebene werden alle Elemente der Anwendungsoberfläche (α^O) über alle Dialoge hinweg berücksichtigt. Ist der Dialog nicht Teil des Merkmals wird die Zugehörigkeit zu einem Dialog auch nicht weiter in die Berechnung mit einbezogen.

$$\begin{aligned} \gamma(\pi) &\rightarrow \{\epsilon_1, \epsilon_2, \dots, \epsilon_n\}, \forall \epsilon : \pi(\epsilon) = \pi \\ &\rightarrow \pi(\epsilon_1) = \pi(\epsilon_2) = \dots = \pi(\epsilon_n) \end{aligned} \quad (5.10)$$

$$\begin{aligned} \gamma(\pi) &\rightarrow \{\delta_1, \delta_2, \dots, \delta_n\}, \forall \delta : \pi(\delta) = \pi \\ &\rightarrow \pi(\delta_1) = \pi(\delta_2) = \dots = \pi(\delta_n) \end{aligned} \quad (5.11)$$

So erhält man für jede Ausprägung des Merkmals eine Gruppe von Elementen (oder Dialogen), die diese Ausprägung erfüllen. Die Elemente einer Gruppe sind somit *bezüglich des Merkmals identisch*.

5.4.4.2 Schritt 2: Berechnung der Transformationen

Für jede homogene Merkmalsgruppe werden nun die korrespondierenden Analyseelemente auf der Variantenplattform herangezogen und die Werte der Merkmale für jedes einzelne Element berechnet. Auf diese Weise kann für jedes Analyseelement eine Transformationsgleichung erstellt werden als die Abbildung des Merkmalstupels im Original auf ein transformiertes Merkmalstupel in der Variante. Dieser Transformationsvektor θ wird hierbei in einzelne Transformationen t_n für die Bestandteile des Merkmals zerlegt.

$$\epsilon^O \Rightarrow \epsilon^V \quad (5.12)$$

$$\pi(\epsilon^O) \Rightarrow \pi(\epsilon^V) \quad (5.13)$$

$$(p_1, p_2, \dots, p_n) \Rightarrow \theta(p_1, p_2, \dots, p_n) \quad (5.14)$$

$$\Rightarrow (t_1 p_1, t_2 p_2, \dots, t_n p_n) \quad (5.15)$$

5.4.4.3 Schritt 3: Berechnung der Konsistenz der Abbildung

Da die Werte (p_1, p_2, \dots, p_n) innerhalb einer Gruppe identisch sind, kann die Abbildung eines Elements in einer Gruppe also auch allein durch die Transformation θ beschrieben werden. Abhängig von der Art des Merkmals (\mathbb{K} , \mathbb{O} oder \mathbb{I}) kann die Transformation nun in verschiedener Form beschrieben werden.

Kategoriale Merkmale: Zur Vereinfachung wird an dieser Stelle von einem kategorialen Merkmal (\mathbb{K}) ausgegangen, so dass als Operation lediglich das Vergleichsurteil $\theta_a == \theta_b, \in \{true, false\}$ möglich ist.

Besteht nun eine merkmalshomogene Gruppe γ aus einer Anzahl von n Elementen $\{\epsilon_1, \epsilon_2, \dots, \epsilon_n\}$ werden diese einzelnen Elemente durch n Transformationen $\{\theta_1, \theta_2, \dots, \theta_n\}$ auf die Varianten abgebildet.

Die Wahrscheinlichkeit einer bestimmten Abbildung entspricht damit der bedingten Wahrscheinlichkeit einer Merkmalskombination in der Variante unter der Voraussetzung einer bestimmten Merkmalskombination im Original.

$$p(\pi(\epsilon^V) | \pi(\epsilon^O)) = \frac{p(\pi(\epsilon^V) \cap \pi(\epsilon^O))}{\pi(\epsilon^O)} \quad (5.16)$$

$$= p(\theta) \quad (5.17)$$

Die Wahrscheinlichkeit einer Abbildung kann somit als einfache Wahrscheinlichkeit der Abbildung bezüglich aller existierender Abbildungen in der Gruppe definiert werden. Die einfache Wahrscheinlichkeit berechnet sich als Häufigkeit h der *häufigsten* Abbildung θ_{max} in Relation zu allen n Abbildungen der Gruppe.

$$p(\theta) = \frac{h(\theta_{max})}{n} (h(\theta_{max}) \max h(\theta_{1...n})) \quad (5.18)$$

Die Konsistenz C der Abbildung einer merkmalshomogenen Gruppe ist nun definiert durch das arithmetische Mittel der Einzelwahrscheinlichkeiten der Gruppe.

$$C_\gamma = \frac{\sum_0^m p(\theta_m)}{m} \quad (5.19)$$

Wobei m die Anzahl der verschiedenen Transformationen in einer merkmalshomogenen Gruppe bezeichnet. Ist die Abbildung in einer Gruppe sehr inkonsistent, bedeutet dies, dass viele verschiedene Transformationen in einer Gruppe angewendet werden. Die Konsistenz der Gruppe sinkt: $\lim_{m \rightarrow n} p(\theta) = 1/n$. Werden dagegen alle Analyseelemente derselben Transformation unterzogen, steigt das Konsistenzmaß der Gruppe an: $\lim_{m \rightarrow 1} p(\theta) = 1$. Ein Konsistenzwert nahe 1 steht somit für eine hohe Konsistenz und ein Konsistenzwert gegen $1/n$ weist auf eine sehr inkonsistente Abbildung hin.

Kontinuierliche Merkmale: Während kategoriale (\mathbb{K}) und ordinale (\mathbb{O}) Merkmale im Umfang der mathematisch möglichen Aussagen auf einfach Vergleichsaussagen reduziert sind, erlauben intervallskalierte (\mathbb{I}) Merkmale die Anwendung kontinuierlicher quantitativer Aussagen. In Abschnitt 5.4.3.2 wurde hierzu das varianzanalytische Konsistenzmaß diskutiert. Die Vor- und Nachteile der stochastischen vs. der varianzanalytischen Methode wurde ebenfalls dort diskutiert.

In Gleichung 5.15 wird eine Transformation (θ) als Kombination der Einzeltransformationen (t_1, t_2, \dots, t_n) definiert. Jede Einzeltransformation wird auf ein Teilmerkmal angewendet. Für eine merkmals-homogene Gruppe gilt damit, dass für jedes Element der Gruppe ein Vektor aus Einzeltransformationen angewendet wird. Innerhalb einer Gruppe streuen die Werte der Transformationen für jedes Teilmerkmal mit der Varianz σ^2 die sich nach Gleichung 5.1 wie folgt berechnet.

$$\sigma_m^2 = \frac{\sum_{n=0}^n (t_m^i - t_m^-)^2}{n} \quad (5.20)$$

Wobei n für die Anzahl der Elemente in einer Gruppe steht. Analog zu Gleichung 5.19 wird nun die Gruppenvarianz als das arithmetische Mittel der Varianzen der Einzelmerkmale berechnet. Um das varianzanalytische Konsistenzmaß analog zum stochastischen Maß auf 1 zu normieren und so zu polarisieren, dass hohe Werte einer hohen Konsistenz entsprechen wird der Mittelwert der Varianzen invertiert und mit 1 addiert.

$$C_\gamma = 1 / \left(\frac{\sum_0^m k \times \sigma_m^2}{m} + 1 \right) \quad (5.21)$$

Wobei m für die Anzahl der Teilmerkmale steht.

Wie Abbildung 5.19 zeigt, fällt das Konsistenzmaß für kleine Varianzveränderungen überproportional stark ab. Über den Faktor k sollte das Konsistenzmaß so an den Wertebereich des Merkmales angepasst werden, dass der sensitive Bereich in einem perzeptuell relevanten Bereich liegt. Der nicht-lineare Verlauf der angepassten Konsistenzfunktion ist motiviert durch Erkenntnisse aus der Physiologie [KB02] wo gezeigt wurde, dass die Empfindlichkeit der menschlichen Wahrnehmung für die Erkennung von Differenzen in größeren Wertebereichen schlechter ist als in niedrigen.

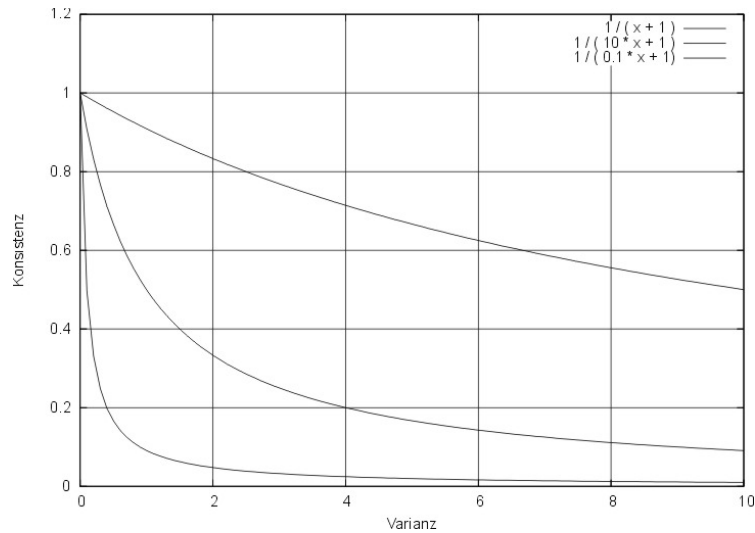


Abbildung 5.19: Der Verlauf des Konsistenzmaßes für verschiedenen Varianzen.

5.4.4.4 Schritt 4: Auswertung und Aufbereitung

Kumulierte Anwendungskonsistenz: Die Frage der sinnvollen Kumulierung der einzelnen Gruppenkonsistenzen hängt in erster Linie vom Verwendungszweck ab und ist in verschiedenen Arten möglich.

Die umfassendste Weise der Kummulierung stellt das anwendungsweite Konsistenzmaß für die Abbildung der Originaldarstellung auf die Variante dar. Hierfür bietet sich das globale arithmetische Mittel über alle Gruppen an:

$$C_{\alpha} = \frac{\sum_0^n C_n^{\gamma}}{n} \quad (5.22)$$

Analyse auf Gruppenebene: Betrachtet man die Elementgruppen, rekrutieren sich die Mitglieder einer Gruppe aus allen Elementen, die im Original identisch bezüglich des Merkmals sind. Konsistent abgebildet, sollte eine Gruppe damit auf der Variante erhalten bleiben.

Die Analyse auf Gruppenebene scheint am geeignetsten, um direkt in den Designprozess eingebunden zu werden. Diese Ebene der Analyse entspricht der Ebene, in welcher der Designer handelt. Dieser verändert meist einzelne Elemente bezüglich ihrer Gestaltung. Entfernt sich das einzelne Element von der konsistenten Abbildung sollte dies direkt während der Veränderung ersichtlich sein (s. Abschnitt 6.2).

Die Rückmeldung sollte dabei möglichst intuitiv in visueller Form dargeboten werden und hierbei auf das aktuell manipulierte Merkmal beschränkt bleiben.

Die Analyse auf Gruppenebene kann zum einen einzelne Elemente gegen den Haupt-Transformationsvektor der Gruppe kontrastieren, zum anderen ist eine Analyse der Gruppenstruktur möglich. Eine Clustering der Transformationsvektoren stellt hierbei eine Möglichkeit dar.

Kreuztabellarische Analyse: Die kreuztabellarische Analyse ist ebenfalls im Sinne der kumulativen Analyse einzusetzen. Sie stellt verschiedene Analyseeinheiten gegenüber. Beispielsweise können so Schwachstellen in der Anpassung durch Gegenüberstellung von Dialog und Elementtyp hervorgehoben werden.

5.4.4.5 Abhängigkeiten der Konsistenz

In Abschnitt 5.4.4.1 wird auf die Bedeutung der Gruppierung von Elementen hingewiesen. Um die Konsistenz der Abbildung bestimmen zu können, muss zunächst die Gruppe von Elementen bestimmt werden, für die die Abbildung im Idealfall gleich sein soll. Diese Gruppe wird nach Gleichung (5.3) über die Ähnlichkeit bezüglich eines Merkmals als Kombination aus einem oder mehreren Einzelmerkmalen definiert.

Hierbei wird klar, dass die Bestimmung der plattformübergreifenden Konsistenz nur dann optimal funktionieren kann, wenn die plattforminterne Konsistenz bereits besteht. Abbildung 5.20 zeigt die wechselseitigen Abhängigkeiten der transformationalen Konsistenz von anderen Formen der Konsistenz.



Abbildung 5.20: Abhängigkeiten der Konsistenz

Plattform Design: Zunächst muss also die Einhaltung von *plattforminternen Designvorschriften und Konventionen* vorausgesetzt werden. Die durchgängige Gestaltung von Interaktionselementen, deren Platzierung und Verwendung muss gegeben sein. Meist wird dies durch die Verwendung von Bibliotheken, den *User Interface Toolkits* 4.2 bereits ausreichend sichergestellt. Zum Teil existieren auch Entwicklungswerkzeuge, die den Designer bei dieser Aufgabe noch zusätzlich unterstützen.

Anwendungskonsistenz: In einem nächsten Schritt ist es notwendig die anwendungsinterne Konsistenz der Originaldarstellung der Anwendung sicherzustellen. Auf diese Weise wird es erst möglich, zusammengehörige Elemente für die plattformübergreifende Konsistenz zu identifizieren und zu gruppieren. Durch die Umsetzung der Anwendungskonsistenz wird versucht, die Gestaltung von Elementen die sich funktional, dialogisch oder semantisch gleichen, auch von der Darstellung gleich zu gestalten.

Verschiedene Merkmale werden also für derartige Elemente auf ähnliche Werte gesetzt. Eben jene Merkmale werden dann zur Berechnung der plattformübergreifenden Konsistenz wiederum zur Gruppierung verwendet. Plattformkonsistenz ist somit eine notwendige Voraussetzung zur effektiven Berechnung der plattformübergreifenden Konsistenz. Werkzeuge wie SHERLOCK(s. Abschnitt 4.4) bieten Methoden an, die Plattformkonsistenz zu verbessern. Aus diesem Grunde sollten Methoden wie die von Mahajan und Shneiderman [MS95] vorgestellten, fester Bestandteil von Werkzeugen zur Messung und Verbesserung plattformübergreifender Konsistenz sein.

Für die im verbleibenden Teil dieses Abschnittes beschriebenen Beispiele wird zur Vereinfachung von plattform-design-konformer und plattformkonsistenten Darstellungen ausgegangen. Der Fokus kann somit auf die Umsetzung der plattformübergreifenden Konsistenzmessung gelegt werden.

5.4.5 Exemplarische Konsistenzberechnung

Die im vorangegangenen Abschnitt beschriebene Herleitung einer plattformübergreifenden Konsistenzmaße soll in diesem Abschnitt anhand einiger Beispiele verdeutlicht werden. Die Beispiele wurden so gewählt, dass sie in Komplexität und Abstraktion aufeinander aufbauen und so die Flexibilität des vorgeschlagenen Maßes veranschaulichen.

1. Die *Konsistenz der Form* zeigt die Anwendung des Konsistenzmaßes auf ein einfaches kategoriales Merkmal anhand der Transformationsstrategie der Substitution.
2. Die *Konsistenz der Position* zeigt die Anwendung des Konsistenzmaßes auf ein einfaches kontinuierliches Merkmal anhand der Transformationsstrategie der Translation.

Eine beispielhafte Anwendung des Konsistenzmaßes in der Entwicklung von plattformübergreifenden Benutzerschnittstellen folgt in Abschnitt 7.2. Die empirische Validierung des Maßes und dessen Effekts auf die Verbesserung der Benutzbarkeit wird weiterhin in Abschnitt 7.3 ausgeführt.

5.4.5.1 Konsistenz der Form

Die Berechnung der Konsistenz ist bei kategorialen Daten nur auf der Ebene der stochastischen Konsistenz möglich. Als ein erstes Beispiel einer Konsistenzberechnung für die Abbildung zwischen zwei Plattformen soll zunächst das Merkmal der Form betrachtet werden. Gleichwohl Ansätze zur abstrakten Beschreibung der Form existieren [Bir33], soll nun der Anschaulichkeit halber von der Form als eindimensionalem kategorialem Kriterium ausgegangen werden. Abbildung 5.21 zeigt die beispielhafte Abbildung der Original-Formen (links) auf die Varianten (rechts) schematisch vereinfacht.

$$\pi = p \ (p \in \{\square, \bigcirc, \triangle, \diamond, \emptyset\}, p \in \mathbb{K}) \quad (5.23)$$

Die Transformation θ zwischen den Darstellungen kann als *Substitution* (s. Abschnitt 5.3.5.2) $t = \text{subst}(p_O, p_V)$ beschrieben werden. In Abbildung 5.21 werden drei Dialoge mit jeweils vier

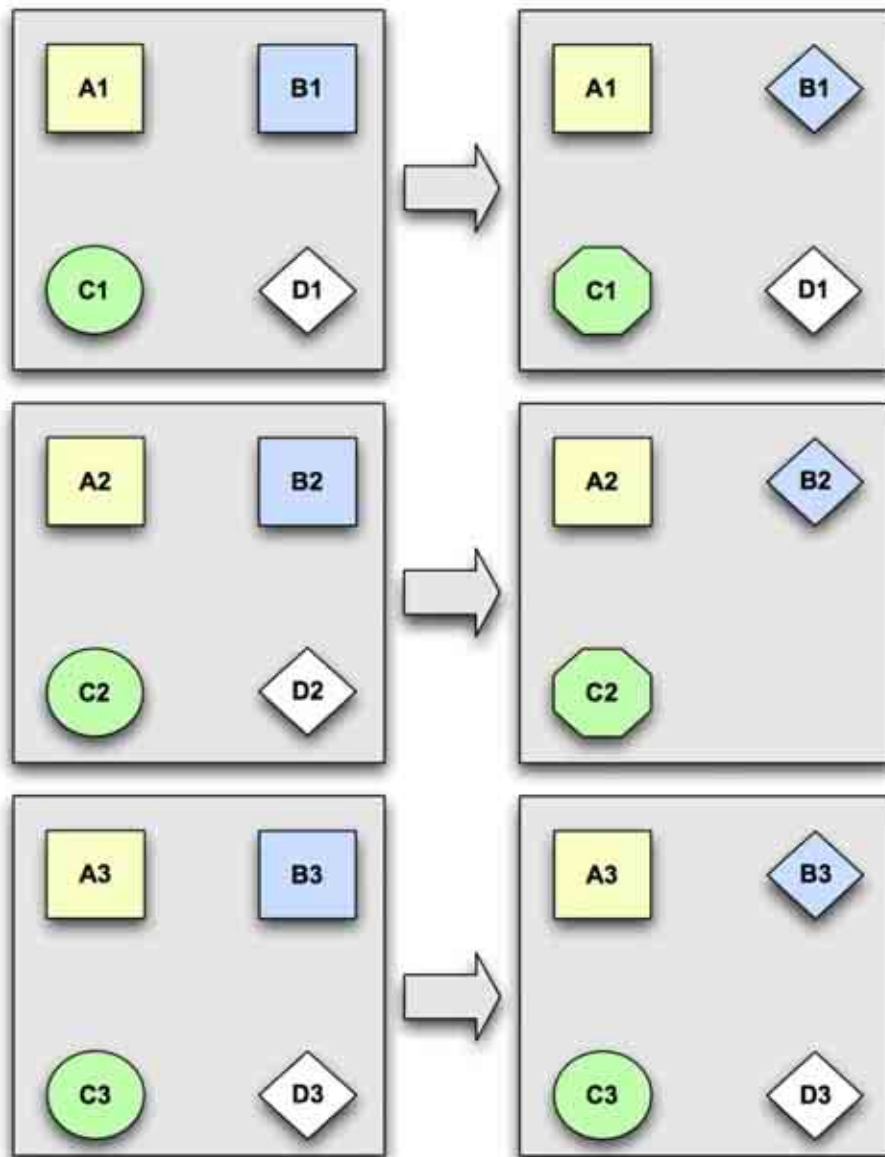


Abbildung 5.21: Schematische Darstellung der Transformation der Form.

Elementen im Original dargestellt. Die Elemente sind im Original alle über alle drei Dialoge identischer Form. Ihre Abbildung auf die Variante hingegen unterscheidet sich.

Tabelle 5.2 zeigt die Auszählung der Häufigkeiten für die verschiedenen Transformationen. Die häufigsten Transformationen sind fett gedruckt.

$subst(p_O, p_V)$		Variante						
		\square	\bigcirc	\triangle	\diamond	\emptyset	Σ	p
Original	$\square = \{A, B\}$	3	0	0	3	0	6	3/6
	$\bigcirc = \{C\}$	0	1	2	0	0	3	2/3
	$\triangle = \{\emptyset\}$	0	0	0	0	0	0	0
	$\diamond = \{D\}$	0	0	0	2	1	3	2/3
	Σ	3	1	2	5	1	12	$\bar{p} = 0,61$

Tabelle 5.2: Häufigkeitsverteilung der Abbildung der Formen

- Das Element $A(\square)$ wird stets auf ein Element der Form \square abgebildet. Die Abbildung dieses Elementes ist also an sich konsistent. Jedoch aufgrund der merkmalsbezogenen Gruppenbildung wird A zusammen mit den Elementen B betrachtet da beide dieselbe Form besitzen. Sie sind somit bezüglich des Merkmals (Form) identisch.
- Das Element $B(\square)$ wird in allen Fällen auf die Form \diamond abgebildet. Auch diese Abbildung ist in sich konsistent. Doch wird die Abbildung zusammen mit dem Element A betrachtet und wird somit inkonsistent. Die Wahrscheinlichkeit $p(\diamond|\square) = p(\square|\square) = 0.5$.
- Das Element $C(\bigcirc)$ wird in zwei Fällen auf \triangle und in einem Fall auf \bigcirc abgebildet. Die Abbildung ist damit inkonsistent da die Wahrscheinlichkeit der häufigeren Darstellung $p(\triangle|\bigcirc) = (2/3)$.
- Das Element $D(\diamond)$ wird in zwei Fällen auf ein \diamond abgebildet, in einem Fall findet eine Auslassung (oder $subst(\diamond, \emptyset)$ statt. Die Konsistenz für diese Gruppe ist damit dieselbe wie bei Element C , $p(\diamond|\diamond) = (2/3)$.

Nach Gleichung 5.22 wird die stochastische plattformübergreifende Konsistenz der gesamten Anwendung berechnet als das arithmetische Mittel der Gruppenkonsistenzen $C_\alpha = \bar{p} = \frac{1/2+2/3+2/3}{3} = 0,61$. Dieser geringe Wert kann dahingehend interpretiert werden, dass bezüglich der Konsistenz der Form noch Potential zur Verbesserung vorhanden ist. Die Gruppenkonsistenzen zeigen hierbei insbesondere bei der \square -Gruppe eine große Uneinheitlichkeit auf.

5.4.5.2 Konsistenz der Position

Die Berechnung der Konsistenz ist bei intervallskalierten Daten auch auf der Ebene der varianz-analytischen Konsistenz möglich. Es soll daher nun die Abbildung zwischen zwei Plattformen soll bezüglich des Merkmals der Position betrachtet werden. Die Position besteht im zweidimensionalen Raum aus einem Vektor mit zwei Werten. In Abbildung 5.22 wird anhand von drei Dialogpaaren die Transformation der Position von drei Elementen exemplarisch aufgezeigt.

Die Transformation θ zwischen den Darstellungen kann als *Translation* (s. Abschnitt 5.3.3.1) $t = trans(p_O, p_V)$ beschrieben werden. In Abbildung 5.22 werden drei Dialoge mit jeweils drei Elementen dargestellt. Die Elemente werden über eine Translation vom Original (links) in die Variante (rechts) abgebildet.

Die Gruppierung der Elemente im Original ist eindeutig, da diese identische Positionen haben. Sie wird in der Abbildung über die Benennung ausgedrückt. Die Position wird vereinfacht normalisiert beschrieben, so dass die Breite und Höhe des Dialoges jeweils mit 1.0 gleichgesetzt wird. Der Nullpunkt wird in der linken unteren Ecke angenommen.

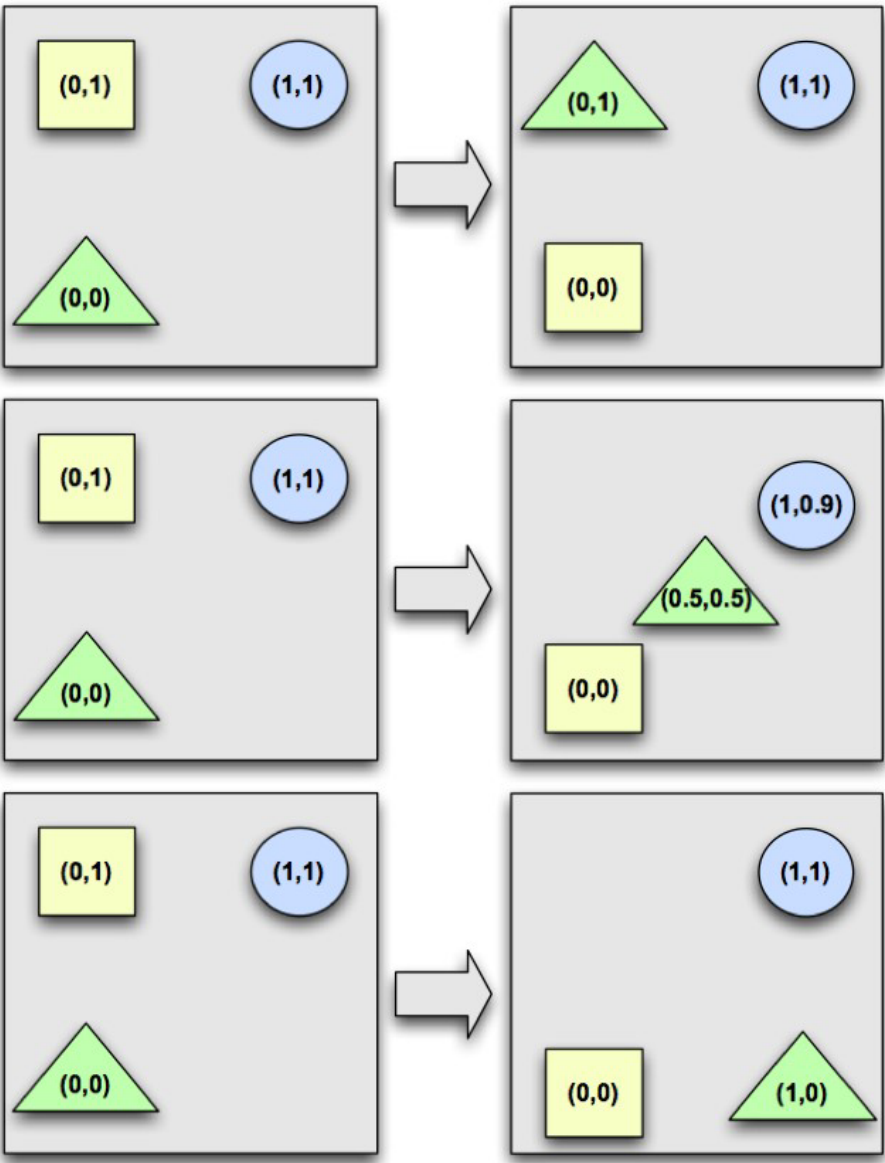


Abbildung 5.22: Schematische Darstellung der Transformation der Position.

$trans(p_O, p_V)$		Reihe				
		Oben	Mitte	Unten	S_{err}^2	$C(k = 2)$
Element	□	(0, -1)	(0, -1)	(0, -1)	(0, 0)	1.00
	○	(0, 0)	(0, -0.1)	(0, 0)	(0, 0.006)	0.99
	△	(0, 1)	(0.5, 0.5)	(1, 0)	(0.5, 0.5)	0.67

Tabelle 5.3: Berechnung der Varianzen und Konsistenz der Translation.

Tabelle 5.3 zeigt die Werte für die Translation der einzelnen Elemente.

Die transformationale Konsistenz wird nun varianzanalytisch aus der Abweichungsquadratsumme (S_{err}^2) berechnet. Diese Berechnung erfolgt gesondert für jedes Merkmal als die quadrierte Abweichung vom Mittelwert. Entsprechend Gleichung 5.21 wird dann das Invers des Mittelwerts der Varianzen plus 1 als Konsistenzmaß erhoben. Im vorliegenden Beispiel wurde der Kontrast des Varianzmaßes mit der Konstanten $k = 2$ angehoben um die relativ geringe Bedeutung der kleinen Abweichung bei \bigcirc gegenüber der starken Abweichung bei \triangle hervorzuheben.

- Das Element \square ist transformational konsistent, da es stets um den Transformationsvektor $(0, -1)$ verschoben wird. Die Varianz der Verschiebung ist somit sowohl auf der x als auch auf der y Achse gleich 0. Die Translation wird damit konsistent angewendet und dies kann mit Hilfe des transformationalen Konsistenzmaßes quantifiziert werden.
- Das Element \bigcirc wird kaum verschoben. Der Translationsvektor beträgt in zwei von drei Fällen $(0, 0)$. In einem Fall ist eine leichte Abweichung um -0.1 beobachtbar, was einer zufälligen Fehlplatzierung entsprechen könnte. Diese Abweichung resultiert in einer leichten Inkonsistenz ($C(k = 2) = 0.99$), deren kleiner Betrag die geringe Relevanz widerspiegelt. Dennoch ist das Konsistenzmaß in der Lage die Abweichung abzubilden und diese dadurch dem Designer anzuzeigen.
- Das Element \triangle wird in jedem Fall auf eine andere Position abgebildet. Die Inkonsistenz dieser Abbildung ist augenfällig. Das Konsistenzmaß reflektiert die starke Abweichung durch eine relativ geringe Konsistenz von 0.67. Anhand der Konstante $k = 2$ wird der Konsistenzwert auf ein relevantes Maß reduziert. Diese Anpassung muss für verschiedene Aspekte oder Fragestellungen optimiert werden.

Die berechneten Gruppenkonsistenzen geben deutlich Auskunft über die Problemfälle dieser Abbildung. Ein Designer könnte auf dieser Weise direkt auf Inkonsistenzen in seiner Designanpassung hingewiesen werden, und die Möglichkeit erhalten, diese direkt zu korrigieren. Der globale Mittelwert der Konsistenz von $\bar{C} = 0,89$ kann als kumulativer Konsistenzwert genutzt werden und zeigt ebenfalls Probleme im Design auf ohne diese detailliert zu benennen.

Dieses Beispiel hat gezeigt, dass die varianzanalytische Konsistenzberechnung eine detaillierte quantitative Beschreibung der Konsistenz verschiedener Elemente einer Darstellung ermöglicht. Dieses Maß lässt sich durch die Wahl der Konstanten k beeinflussen und sollte für verschiedene Aspekte normiert werden. Hierzu bietet sich es an, wie im vorliegenden Fall, auch die Maße der Aspekte normalisiert einzubeziehen.

5.5 Zusammenfassung

Mit dem Referenzmodell der plattformunabhängigen Entwicklung von User Interfaces wurde in diesem Abschnitt ein Ansatz vorgestellt und theoretisch motiviert, wie die in den Abschnitten 2.3 und 2.4 identifizierten Anforderungen durch eine benutzerzentriertes Vorgehensmodell erfüllt werden können. Das Vorgehen orientiert sich in seiner grundlegenden Struktur an Mayhews *Usability Engineering Lifecycle* Modell und gliedert den Entwicklungsprozess in drei, aufeinander aufbauenden Stufen. Insbesondere die Auftrennung der eigentlichen Entwicklungsarbeit von der konzeptionellen Sammlung der Anforderungen hebt das Modell von rein modellorientierten Ansätzen ab. In der zentralen Entwicklungsphase wird zwischen verschiedenen Ebenen der Darstellung unterschieden, ein Prinzip, welches als Prinzip der Sequenziellen Anpassung bezeichnet wurde. Die Anwendung der geordneten Entwicklungspfade fordert eine Unterscheidung in primäre und sekundäre Entwicklungspfade, ein Ansatz der technisch und ergonomisch motiviert ist und sich von den sog. *multi-path* Methoden absetzt. Die zentrale Annahme des Entwicklungsmodells leitet seine Forderungen aus den Untersuchungen zur plattformübergreifenden Konsistenz und dem menschlichen Wissenserwerb und Transfer (s. Abschnitte 2.2, 3.2, 3.3, und 3.1) ab, und wurde als Transformationsparadigma bezeichnet. Zentrale Aussage ist, dass der Benutzer eines plattformübergreifenden Systems, die verschiedenen Darstellungen auf verschiedenen Endgeräten als Ableitungen, bzw. Transformationen des ursprünglichen Interfaces wahrnimmt. Diese Darstellung unterscheidet sich in großem Maße von der

Sicht des Entwicklers, welcher als tatsächliches Vorgehen die unabhängige Erzeugung aus einem gemeinsamen abstrakten Modell kennt. Dieses konzeptuelle Modell differiert also erheblich von des Benutzers mentalem Modell. Der Entwicklungsprozess muss dieser Tatsache Rechnung tragen und zum einen durch die Gestaltung von Abläufen, zum anderen durch werkzeuggestützte Methoden der Evaluation den Entwickler unterstützen. Das Referenzmodell der plattformübergreifenden Entwicklung von User Interfaces bietet ein konzeptionelles Rahmenwerk zur Gestaltung und Entwicklung einer Entwicklungsplattform. Im nächsten Abschnitt wird eine Systemarchitektur für eine solche Entwicklungsplattform ausführlich dargestellt und erläutert werden. Diese Architektur ist in dem Maße universell als dass sie sowohl für die Entwicklung von Benutzerschnittstellen, als auch als Laufzeitumgebung zu deren Darstellung genutzt werden können.

Weiterhin dient das Referenzmodell als integrierender Rahmen zur Einordnung wichtiger Forschungsfragen der Domäne der Mensch-Computer Interaktion. Gleichwohl die Grundannahmen des Modells auf allgemein anerkannten Konzepten der experimentellen Psychologie und Erkenntnissen der Forschung der Mensch-Maschine Interaktion aufbauen, so ist dennoch die Gültigkeit mancher Hypothesen für den Bereich der plattformübergreifenden Interaktion nicht, oder nur unzureichend nachgewiesen. Aspekte, wie die Dominanz der gewohnten Plattform und deren Auswirkung auf den Interaktionsstil oder die Implikationen des Transformationsparadigmas können in dieser Arbeit nur zum Teil, in keinem Falle jedoch erschöpfend geklärt werden. In den Abschnitten 6.2 und 7 wird der empirische Nachweis dieser Thesen ausführlicher behandelt werden.

Das transformationale Konsistenzmaß ermöglicht die Berechnung eines numerischen Konsistenzwertes. Konsistenz wird als einer der wichtigsten Aspekte der Benutzbarkeit von User Interfaces gewertet (s. Abschnitt 3.3) und somit kann mittels dieses Maßes die Benutzbarkeit eines User Interfaces antizipiert werden. Veränderungen in der Gestaltung eines Designs können so unmittelbar in einer Bewertung der Konsistenz reflektiert werden. Methoden der konsistenten Entwicklung können auf diesem Maß unmittelbar aufbauen und eine neue Qualität der Unterstützung für Entwickler bieten. Dieses Maß ist bezüglich des zu bewertenden Maßes offen gehalten so dass prinzipiell die Evaluierung verschiedener Aspekte der Designs, auch in einer multimodalen Anwendung möglich ist. Insbesondere die Tatsache dass auf diese Weise die plattformübergreifende Konsistenz unter Berücksichtigung plattformspezifischer Besonderheiten möglich ist, stellt eine wichtige Eigenschaft des transformationalen Maßes dar. Die Umsetzung und Integration des Konsistenzmaßes in der graphisch-interaktiven Entwicklung von User Interfaces wird im nächsten Abschnitt für die plattforminterne und plattformübergreifende Anwendung diskutiert werden. Die Validierung des Ansatzes erfolgt in den Abschnitten 7.2 und 7.3.

6 Realisierung der Entwicklungskonzepte

Die Realisierung der Konzepte zur Unterstützung der Entwicklung plattformübergreifender Benutzerschnittstellen, wie sie im vorangegangenen Abschnitt 5 vorgestellt wurden, basiert auf zwei wesentlichen Beiträgen: eine Darstellungsplattform für plattformübergreifende Benutzerschnittstellen, sowie Methoden zur Unterstützung des Designprozesses basierend auf dem Konzept der transformationalen Konsistenz.

Darstellungsplattform Ein System zur Darstellung und vereinfachten Entwicklung plattformübergreifender Benutzerschnittstellen. Das System versteckt Anpassungsprozesse und plattformspezifischen Anforderungen weitestgehend in einer toolkitorientierten Entwicklungsschnittstelle (s. Abschnitt 6.1).

Methoden der Design-Unterstützung Eine Anzahl von Entwicklungsmethoden und deren beispielhafte Umsetzung. Die Methoden basieren auf dem Transformationskonzept und haben zum Ziel, die Einbindung des Konsistenzmaßes in den graphisch-interaktiven Designprozess zu ermöglichen (s. Abschnitt 6.2).

Sowohl die Darstellungsplattform als auch die Methoden der Design-Unterstützung haben zum Ziel, die Entwicklung von Benutzerschnittstellen für verschiedene Endgeräte, zu verbessern. Diese Umsetzungen stellen eine Anwendung des *Referenzmodells der plattformübergreifenden Entwicklung* (Abschnitt 5.2) dar und basieren auf den Konzepten der *benutzerzentrierten Entwicklung* [May99] (s. Abschnitt 5.1). Die Betrachtung des Entwicklers als Benutzer der Entwicklungswerkzeuge spielt hierbei eine wesentliche Rolle bei der Definition der Rollen in dem System.

Insbesondere die Methoden der Design-Unterstützung greifen einen wichtigen Bestandteil des Entwicklungsmodells, das Transformationsparadigma (Abschnitt 5.2.3) und die daraus abgeleiteten Methoden zur Erfassung der transformationalen Konsistenz (Abschnitt 5.4) auf. Gemäß der postulierten fachübergreifenden Vorgehensweise konnten so kognitions- und wahrnehmungspsychologische Aspekte, wie das Konzept der mentalen Modelle [JL83, Nor83], in den Softwareentwicklungsprozess eingebracht werden und benutzerzentrierte Unterstützungsmethoden zur Verbesserung dieses Prozesses definiert werden.

6.1 Eine Darstellungsplattform

Aufbauend auf dem Referenzmodell der plattformübergreifenden Entwicklung (Abschnitt 5.2) wird in diesem Abschnitt der Entwurf einer Systemarchitektur zur Darstellung und Entwicklung plattformübergreifender Benutzerschnittstellen vorgestellt.

Zunächst werden allgemeine technische Aspekte der Plattform diskutiert (Abschnitt 6.1.1). Überlegungen zur konzeptionellen Architektur basierend auf dem ARCH-Modell und dessen Verknüpfung mit dem Referenzmodell werden diskutiert. Die Verwendung eines Standard des *World Wide Web Consortium (W3C)* zur Spezifikation elektronischer Formulare *XForms* und dessen Erweiterung zu der *eXtensible User Interface (XUI)* Sprache wird kurz angerissen, ebenso wie die Verwendung der *Microsoft .Net* Plattform.

Daraufhin werden in Abschnitt 6.1.2 verschiedene Einsatzszenarien, neben der plattformübergreifenden Nutzung, aufgezeigt. Eine detaillierte Diskussion der Aspekte der Anwendungsintegration durch eine mehrstufig abstrahierte Anwendungsprogrammierungsschnittstelle (API) und der verschiedenen Ebenen der Konkretisierung der Benutzerschnittstelle aus einer abstrakten Präsentation werden in den Abschnitten (6.1.3 bis 6.1.7) erläutert.

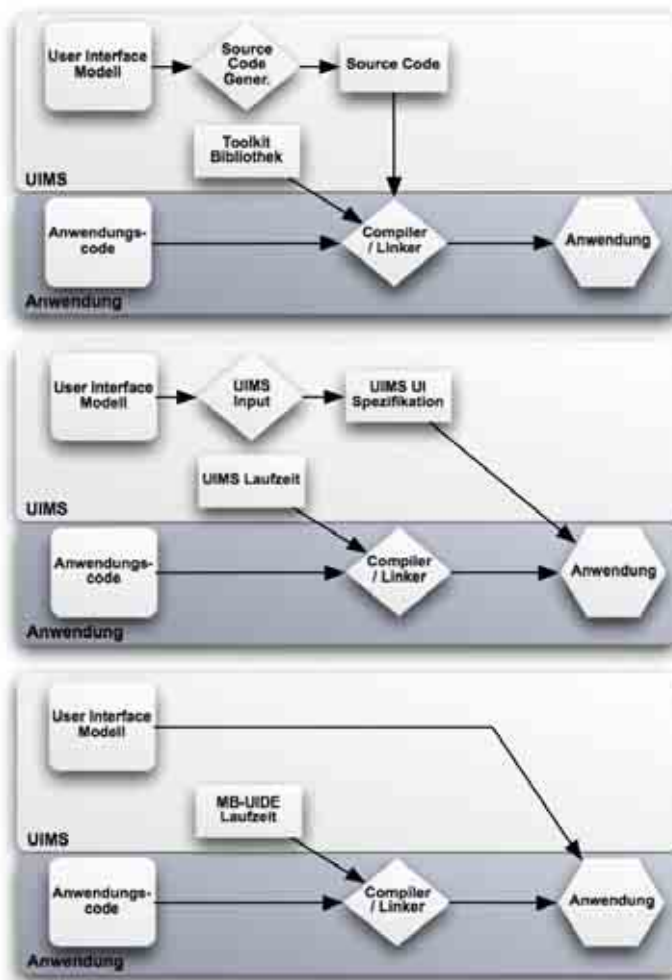


Abbildung 6.1: Drei Ansätze der Realisierung von UIMS Laufzeitarchitekturen nach [Pin00].

6.1.1 Übersicht

Der Zeitpunkt der Erzeugung der Darstellung ist ein wichtiges Kriterium plattformadaptiver Systeme [TCC04, Pin00]. Prinzipiell kann zwischen einer Erzeugung zur *Designzeit* und zur *Laufzeit* unterschieden werden. Abbildung 6.1 zeigt diese beiden Ansätze nach [Pin00]. Die Erzeugung zur Laufzeit wird hier nochmals zwischen der Möglichkeit der Vorverarbeitung durch die Übersetzung der Modelle in eine Zwischenspezifikation und der kompletten Interpretation der Modelle zur Laufzeit unterschieden.

Wird eine Darstellung zur Designzeit erzeugt, resultiert dies meist in kompiliertem Code, der für eine kontextsensitive Anpassung zur Laufzeit nicht oder nur bedingt geeignet ist. Die Dimensionen und Ausprägungen der Anpassung sind in der Regel auf die, vom Designer berücksichtigten Maße beschränkt. Wird der die Darstellung zur Laufzeit erzeugt, kann diese in weit größerem Maße auf den aktuellen Kontext angepasst werden. In der Regel ist diese Form der Darstellung allerdings deutlich komplexer und damit aufwändiger als die vorgefertigte. Eine Kombination zwischen Design- und Laufzeiterzeugung ist ebenfalls möglich.

Die Darstellungsplattform unterstützt prinzipiell beide Ansätze. Allerdings ist aufgrund der größeren Flexibilität der Schwerpunkt auf eine Erzeugung zur Laufzeit gelegt worden. Die weitere Diskussion wird sich aus diesem Grund auf diese Strategie konzentrieren. Allerdings lassen sich hieraus prinzipiell Möglichkeiten der Kompilation abgeschlossener oder teiladaptiver Anwendungsoberflächen ableiten, indem die modular beschriebenen Prozesse integriert werden.

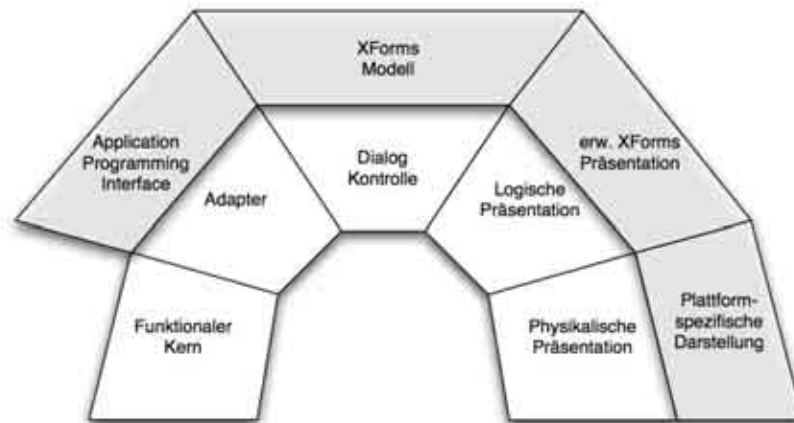


Abbildung 6.2: Das ARCH Modell und dessen Adaption auf die Architektur der Entwicklungsplattform nach [She92].

Die Darstellungsplattform stellt ein UIMS (s. Abschnitt 4.3) dar, in welchem zur deklarativen Spezifikation der Dialoge die *XForms*-Erweiterung eingesetzt wird. Bevor auf die einzelnen logischen Komponenten des Systems im detailliert eingegangen wird, sollen zunächst einige allgemeine Eigenschaften erklärt werden. Diese werden stets in direkten Bezug zu den, in Abschnitt 2 beschriebenen Anforderungen gesetzt.

6.1.1.1 Das ARCH-Modell

Die Darstellungsplattform ist eine Anwendung des ARCH-Modells nach Bass *et al.*[She92]. Abbildung 6.2 zeigt, wie sich die Ebenen des Darstellungsmodells auf das ARCH-Modell übertragen lassen. Auf der Anwendungsseite steht im Fall der Darstellungsplattform eine API, die den Zugriff der Anwendung auf das Darstellungssystem ermöglicht. Auf der Benutzerseite steht die plattformspezifische Darstellung. Dazwischen wird über eine Abstraktionsschicht, dem *Abstrakten User Interface* vermittelt.

Thevenin *et al.*[TCC04] betrachten das ARCH-Modell in Bezug auf dessen Eignung für die Entwicklung plattformübergreifender Benutzerschnittstellen. Das ARCH-Modell teilt die Benutzerschnittstelle in fünf Ebenen auf. Zuunterst stellt der *funktionale Kern* die Anwendungslogik dar. Die *Anpassungsschicht* zwischen Anwendungslogik und dem der Benutzungsoberfläche wird von der *Dialog-Kontrolle* übernommen. Diese kontrolliert die Übersetzung der Funktionen in Elemente der Oberfläche und steuert den Ablauf. Die *logische Präsentation* repräsentiert die Interaktoren welche sich in der *physikalischen Präsentation* manifestieren.

Anwendungskern (functional core, FC) stellt die Anwendungslogik selbst dar. Moderne Oberflächen Architekturen [Gre85, She92, Sha90, Pha00, DKMR03] versuchen meist, eine Abstraktion von der eigentlichen Anwendung und der Benutzerschnittstelle einzuführen, um so die Austauschbarkeit und Unabhängigkeit von der Darstellung zu gewährleisten. Daher ist der funktionale Kern selbst nur indirekt mit der Darstellung selbst über eine Schnittstelle verbunden.

Anpassungsschicht (functional core adapter, FCA) Stellt die Schnittstelle dar. Sie stellt zum einen die Funktionen des FC für die Benutzerschnittstelle zur Verfügung und ermöglicht zum anderen den Zugriff auf die Darstellung für den FC. Im heutigen Entwicklungsprozess wird diese Schicht in der Regel durch einen User Interface Toolkit und dessen API realisiert. Ein wichtiger Aspekt der Darstellungsplattform ist es, eine solche API zur Verfügung zu stellen und so die komplexen Prozesse der Anpassung hinter In dem hier vorgestellten Ansatz wird versucht, den Entwickler durch eine solche API zu unterstützen. Die komplexen Prozesse und Strukturen des Darstellungsprozesses werden für den Entwickler transparent gehandhabt. Dieser kann die Entwicklung über eine API realisieren, welche in weiten Teilen der

Standard Toolkit API entspricht (in der konkreten Implementierung ist dies die *Microsoft .Net WinForms* API).

Dialog-Kontrolle (dialog control, DC) Hier wird der Zustand des Dialoges, die Abhängigkeiten und das Datenmodell repräsentiert. Diese Ebene ist darstellungsunabhängig und repräsentiert die inhaltlichen Aspekte des Dialogs. Auf sie greift die logische Präsentation zurück um die Inhalte und Interaktionen zu regeln. In der hier vorgestellten Realisierung nutzt diese Ebene die XFORMS-Modellkomponente.

Logische Präsentation (logical presentation, LP) Sie greift auf die Ebene der Dialog-Kontrolle zu, um diese in die Präsentation einzubinden. Sie ist nach wie vor darstellungs- und plattformunabhängig und definiert abstrakte Interaktoren. Die Abhängigkeiten und Logik sind in der DC-Ebene definiert. Die Spezifikation abstrakter Elemente ist eine umfangreiche Aufgabe und es existieren bereits einige Ansätze, die diese Aufgabe erfüllen. In der vorliegenden Arbeit wurde, wie in der DC-Ebene, exemplarisch auf den W3C Standard XFORMS zurückgegriffen.

Physikalischen Präsentation (physical presentation, PP) Diese Ebene umfasst die kompletten Darstellungs- und Anpassungsprozesse, wie sie im Referenzmodell definiert wurden. Die zur Darstellung benötigten Gestaltungsattribute müssen für jeden Darstellungs- und Anpassungsschritt überlagert werden. Eine hierfür geeignete Technologie wurde in dem W3C Standard *Cascading Style Sheets (CSS)* [LB99b, LB99a] gefunden.

6.1.1.2 Systemarchitektur

Überträgt man das ARCH-Modell nun in konkrete Systemkomponenten, erhält man eine geschichtete Systemarchitektur. Diese benötigt zur Erzeugung einer Darstellung mehrere Anpassungsschritte. Abbildung 6.3 zeigt die Übersicht der Systemarchitektur. Das System wird in zwei Teilsysteme unterteilt, wobei eines primär die Darstellung einer Oberfläche betrifft während das zweite als Entwicklungskomponente auf diesem aufsetzt. Das Entwicklungssystem setzt auf dem Darstellungssystem auf und beinhaltet Komponenten zur graphischen Gestaltung und Konfiguration. Beide Systeme hängen also voneinander ab, wobei das Darstellungssystem als Untersystem des Entwicklungssystems unabhängig laufen kann. Es kann als alternativer User Interface Toolkit auf eine Plattform eingespielt und dort in Anwendungen eingebunden werden.

6.1.1.3 Das *Microsoft .Net Framework*

Eine konkrete Umsetzung der Plattform wurde im Rahmen dieser Arbeit realisiert und verwendet im wesentlichen Techniken der *Microsoft .Net Plattform* (s. auch Abschnitt 4.2).

Die Entscheidung für diese proprietäre Plattform wurde getroffen da die *.Net Plattform* der *Java Virtual Machine* technisch in weiten Teilen gleicht. So wird auch hier zur Designzeit nur eine plattformunabhängiger Intermediärkode (*Bytecode*) erzeugt, der dann zur Laufzeit auf dem Zielgerät übersetzt wird. Insbesondere der Anwendungskern kann in vielen Fällen auf allen Plattformen ohne Veränderung genutzt werden. Während Sun's Java allerdings die Strategie verfolgt klar getrennt Produkte für verschiedene Plattformen zu vertreiben, stellt das *.Net Framework* weitestgehende Übertragbarkeit her.

Ein weiterer Grund für die Wahl dieser Technologie, war weiterhin die Tatsache, dass Microsoft bei der Konzeption der *.Net Plattform* die Entwicklung verschiedene Endgeräten berücksichtigt. Aus diesem Grund wurden zum einen eine durchgängige API (in reduzierter Schnittmenge) für die plattformübergreifende Entwicklung bereitgestellt. Zum anderen wurde die plattformübergreifende Entwicklung im Rahmen der *Microsoft Visual Studio .Net* Entwicklungsumgebung bereits weitestgehend in den Entwicklungsprozess integriert. So unterstützt das *Visual Studio* heute über Erweiterungen bereits die Entwicklung aller Microsoft Plattformen von Desktop PC, Tablet PC, Windows CE, Pocket PC, und Smartphone.

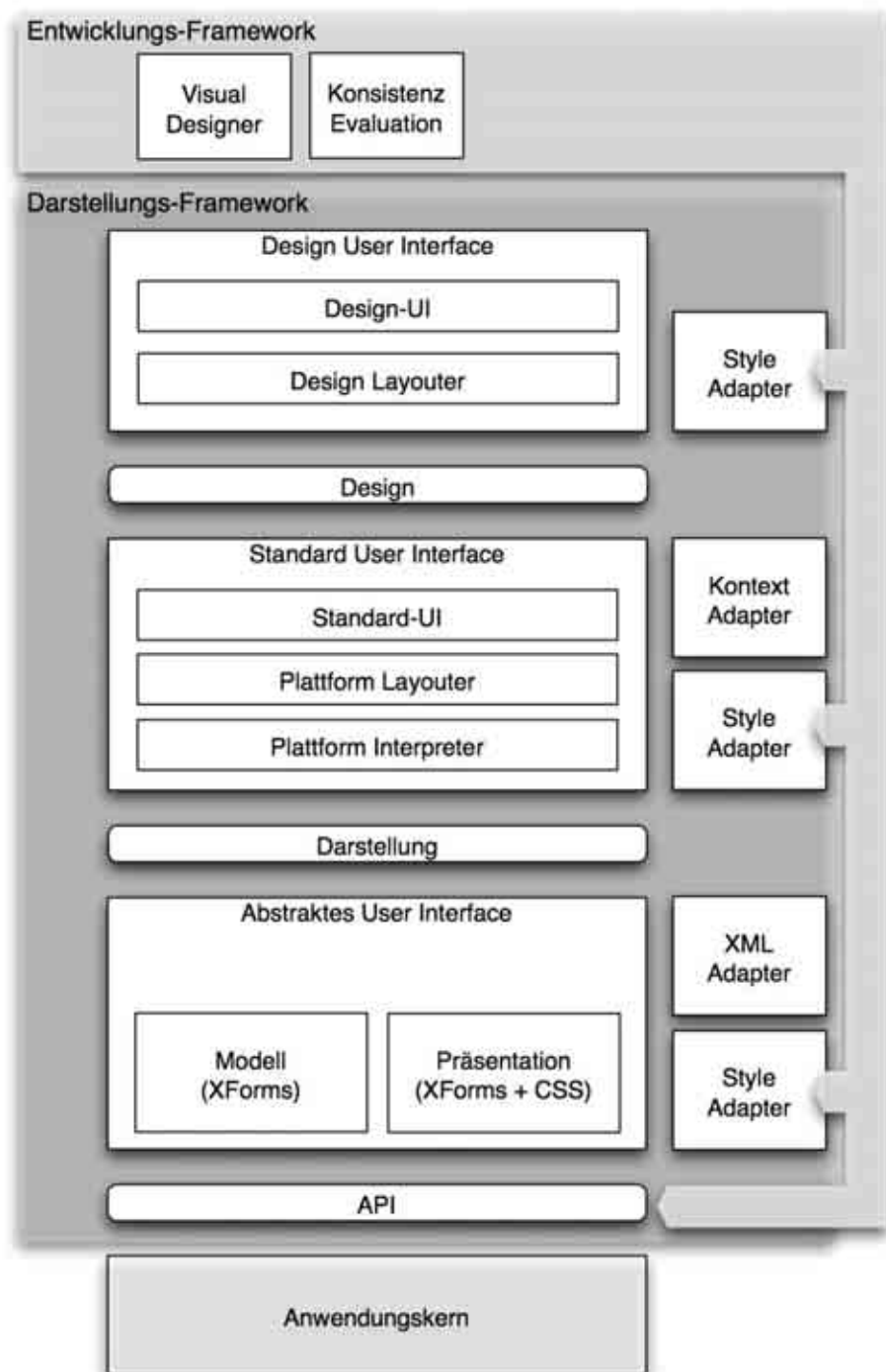


Abbildung 6.3: Übersicht über die Architektur des Darstellungssystems.

Die Integration der mobilen Plattformen wurde in der zweiten Version der .Net Plattform weiter vorangetrieben. Hier werden dieselben Laufzeiten und Projekte für PDA und Smartphones verwendet. Auch die Schnittstellentechnologie *Windows Vista* [Eck06] mit der kanonischen XML-basierten Beschreibungssprache XAML wird einen weiteren Schritt in diese Richtung darstellen.

Über die Entwicklung der Darstellungsplattform konnte so auch Erfahrung bezüglich der Problematiken einer der wenigen existierenden kommerziellen plattformübergreifenden graphischen Entwicklungsumgebung wie *Microsoft Visual Studio* gewonnen werden. Diese Erfahrungen haben im wesentlichen die Ergebnisse der Anforderungsanalyse bestätigt und Schwachstellen des existierenden Systems offenbart, welche mit dem hier vorgestellten System adressiert werden sollen. Insbesondere bei den Konzepten in Abschnitt 6.2 wurden die Erfahrungen aus der Arbeit mit *Visual Studio* eingebracht.

Die wichtigsten Erkenntnisse aus dem Umgang mit *Microsoft Visual Studio .Net* seien an dieser Stelle kurz zusammengefasst:

- Die API hatte im wesentlichen einen kleinsten gemeinsamen Satz an Funktionen, die auf allen Plattformen verfügbar waren. Die Verwendung von Funktionen, die in einer der (kleineren) Plattformen nicht verfügbar waren, führten häufig zum Absturz erst zur Laufzeit, meist ohne aussagekräftige Kommentare.

Mögliche Lösungsansätze sind hier:

- Wahlmöglichkeit eines verlässlichen Kompatibilitätsmodus.
- Verbesserte Debug-Informationen.
- Gutmütige Reaktion des Programms indem diese Funktionsaufrufe ignoriert werden.
- Werkzeug zur einfachen und verlässlichen Kompatibilitätsüberprüfung.

Im Rahmen des hier vorgestellten Ansatzes wurde mit der XAPI (s. Abschnitt 6.1.3 der Weg des Kompatibilitätsmodus gewählt.

- Zwar können in derselben Entwicklungsumgebung Benutzerschnittstellen für alle Plattformen entwickelt werden, pro Projekt ist jedoch nur eine Plattform möglich, so dass eine wirkliche plattformübergreifende Entwicklung nicht möglich ist.

Hierzu notwendig ist zumindest:

- Die synchrone Ansicht aller plattformspezifischen Varianten.
- Die Unterstützung bei der Anpassung.

Im Rahmen dieser Arbeit wurde eine Plattform entwickelt, die aufgrund des modularen Aufbaus konzeptionell die synchrone Darstellung und Gestaltung vorsieht. Ansätze zur Unterstützung bei der Anpassung werden sowohl in diesem, wie in Abschnitt 6.2 beschrieben.

- Auch die plattforminterne Entwicklung von Benutzerschnittstellen kann durch Hilfsmethoden bei der Gestaltung unterstützt werden. Graphische Editoren ermöglichen hierbei einen hohen Grad interaktiver und intuitiver Unterstützung. Hiervon stehen bislang in den meisten kommerziellen Werkzeugen nur wenige zur Verfügung. Ein Beispiel, welches im Rahmen dieser Arbeit insbesondere zum tragen kam, war die Unterstützung bei der Entwicklung konsistenter Darstellungen. Beispielhafte Konzepte hierfür werden in Abschnitt 6.2.2 beschrieben.
- Andere, technische Probleme, die die Verwendung von dynamischen Bibliotheken, mangelnde Unterstützung von XML-Prozessen und die Verteilung der Anwendung betreffen werden in den Arbeiten von Kopp [Kop04], Herold [Her05] und Lanz [Lan05] beschrieben.

6.1.1.4 W3C XFORMS

Die Umsetzung des Darstellungssystems basiert in wesentlichen Teilen auf der W3C Empfehlung der XFORMS [DKMR03, Dub03] (s. dazu auch 4.3.1). XFORMS ist ein Standard für elektronische Formulare zur Datenerfassung, welcher 2003 vom W3C in der Version 1.0 freigegeben wurde. Die Überarbeitung zur Version 1.1 ist derzeit in Arbeit [Boy05]. Dieser Standard soll dazu dienen, die veralteten Formularelemente in HTML abzulösen, da diese von der Funktionalität nicht ausreichen.

Bei XFORMS handelt es sich nicht um ein völlig eigenständiges Format. Formulare in XFORMS müssen in ein anderes Dokument, z. B. ein SVG oder XHTML, eingebettet werden. Man spricht hierbei von der *Host Language*. XFORMS bezieht sich in seiner Spezifikation auf weitere W3C-Standards wie den XML Schema Definitionen [FW04], XML Namespaces [BHL99] oder XPath [CD99].

Die Definition einer XFORMS-Anwendung geschieht immer in zwei Modellen. Diese erfolgt analog zur strikten Trennung von Daten und Präsentation. Die Daten werden im *Modell* als alleinstehendes XML-Dokument definiert. Die Daten können somit durch jeden standardkonformen Verarbeitungsprozess empfangen werden. Die Präsentation im *Interface* definiert die Interaktionselemente und deren Bindung an das Datenmodell über die XML Pfadangaben. Zur Präsentation definiert XFORMS einen eingeschränkten Satz an Bedienelementen, welche direkt in *Host Languages* wie XHTML verwendet werden können. Es können jedoch genauso andere Darstellungen definiert und an das Modell gebunden werden.

XFORMS wurde aus verschiedenen Gründen als Basis für die Beschreibung der Benutzerschnittstelle gewählt. Wie in den Anforderungen bereits erwähnt wurde, ist die Verwendung von Standards ein wichtiger Faktor bei der Entwicklung eines neuen Systems. Standards erlauben die Verwendung existierender Werkzeuge und garantieren die Erhaltung des Wertes einer Anwendung aufgrund der Tatsache dass Standards meist über Jahre stabil bleiben. Neben den XFORMS haben in neuerer Zeit nur zwei alternative Ansätze den Weg der Standardisierung verfolgt UIML [AP99] und V2 als ANSI/INCITS 389ff. [Int05a]. Während UIML einen eigenständigen Weg verfolgt, sieht der ANSI-Standard 389ff. die Verwendung anderer Darstellungssprachen vor und ist damit kompatibel zu XFORMS. Die Standardisierungsbemühungen von UIML treffen lediglich auf geringe Unterstützung wohingegen der ANSI Standard konkret den Kontakt zu Industrieorganisationen wie *European Committee of Domestic Equipment Manufacturers (CECED)* oder *Universal Plug and Play (UPnP)* sucht und damit größere Aussichten auf Erfolg zu haben scheint.

Die Verwendung von XFORMS ermöglicht weiterhin die Darstellung der Benutzerschnittstelle in existierenden XFORMS Browsern (X-Smiles [HV04], FormsPlayer, Sun Star Office, etc.) eine Tatsache, die insbesondere bei verteilten Szenarien von Bedeutung sein kann.

Die *Device Independency Workgroup* des W3C betreibt ebenfalls die einer plattformunabhängigen Spezifikation von Benutzerschnittstellen. Diese soll voraussichtlich in wesentlichen Teilen auf der XForms und XHTML Spezifikation [Boy04] basieren.

6.1.2 Einsatzszenarien

Gemäß den In Abschnitt 2.5 beschriebenen technischen Anforderungen, wurde Wert auf ein skalierbares System gelegt. Die Skalierbarkeit kann am einfachsten durch Modularisierung gewährleistet werden. Werden die einzelnen logischen Module eines Systems physikalisch getrennt eingesetzt, lassen sich Rechenressourcen u. U. auf leistungsfähige Server auslagern, bzw. im Sinne verteilter Dienste dynamisch einbinden und erweitern.

Der modulare Aufbau des Systems ermöglicht daher eine sehr flexible Konfiguration bezüglich der Einsatzszenarien und der Ein- und Ausgabemöglichkeiten. Es ist sowohl der lokale Einsatz als Anwendungsoberfläche im Sinne eines User Interface Toolkits als auch als Integrations- und Interaktionsschicht für multimodale oder serverbasierte sog. *Thin Client* Lösungen denkbar und möglich. Im folgenden wird eine Übersicht über die möglichen Einsatzszenarien des Darstellungssystems gegeben.

6.1.2.1 Lokaler Einsatz

Abbildung 6.4 verdeutlicht den Aufbau des lokalen Einsatzes des Darstellungssystems. Das gesamte System, inklusive Anwendungskern, läuft auf einem Endgerät. Die konkrete Ebene des Systems ist plattformspezifisch implementiert und realisiert die Abbildung in eine plattformspezifische Darstellung.

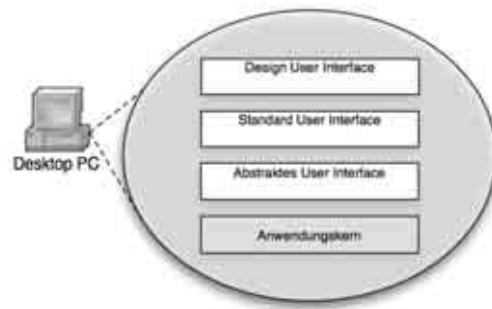


Abbildung 6.4: Lokaler Einsatz auf einer Plattform

Dieses System erlaubt den unabhängigen Betrieb des Darstellungsprozesses auch auf eingeschränkten Geräten, da hier eine praktische Verschmelzung der logischen Ebenen oberhalb der API ermöglicht wird und so Rechenaufwand reduziert wird. Jedoch kann auch die gesamte Plattform auf einem Gerät laufen. Dies ist in der existierenden Implementierung der Fall. Hier werden nur die Darstellungsmodule, abhängig von der Plattform aus dem Laufzeitsystem ermittelt.

6.1.2.2 Multimodaler Einsatz

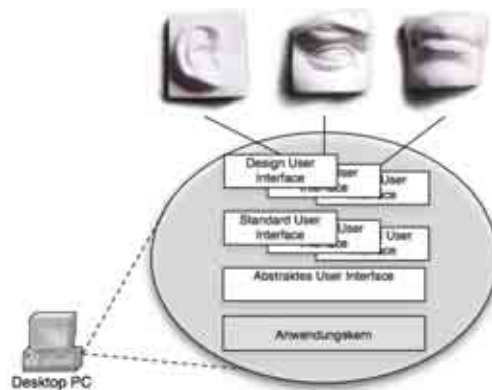


Abbildung 6.5: Lokaler Einsatz auf einer Plattform als multimodale Plattform

Die Abbildung erfolgt auf dem Endgerät in verschiedene multimodale Darstellungen und ermöglicht ebenfalls die Eingaben über verschiedene Modalitäten. Die Konfiguration der Ein- und Ausgaben kann dynamisch kontextsensitiv (s. Abschnitt 7.1) oder gerätespezifisch und vordefiniert erfolgen. Als Integrationsschicht für konkurrierende Eingaben kann die abstrakte Ebene der Dialogkontrolle bzw. die logische Präsentation dienen. Diese hält einen konsistenten Systemzustand aufrecht und löst Konflikte auf. Das System kann lokal (s. Abbildung 6.5) oder über eine Netzwerkverbindung (s.u. *remote*) aufgebaut sein.

6.1.2.3 Netzwerkbasierter Einsatz

Die Auftrennung der Darstellung in verschiedene Ebenen ermöglicht eine physikalische Trennung der einzelnen Komponenten über eine Netzwerkschicht. Zwei mögliche Auftrennungen bieten sich hier an: einmal als integriertes *Thin Client Konzept* (s. Abbildung 6.6) und einmal eine *Client-Server-basierte Lösung* mittels eines XFORMS-fähigen Internet Browsers (s. Abbildung 6.7).

Die *Thin Client Lösung* basiert auf einem ähnlichen Ansatz, wie die multimodale Darstellung (s. o.). Die abstrakte Ebene eignet sich für eine Abtrennung, da hier, wie oben beim multimodalen Einsatz

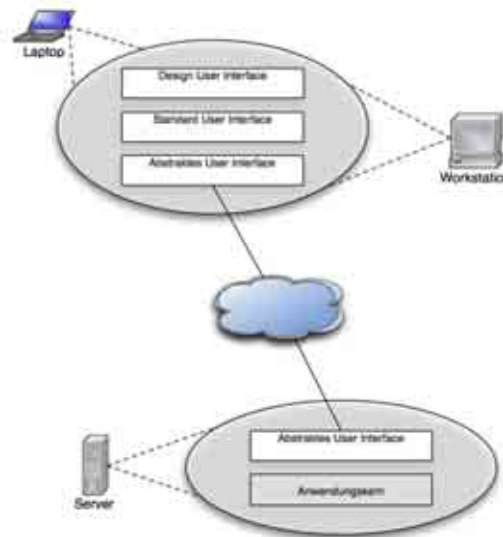


Abbildung 6.6: Einsatz über eine Netzwerkschnittstelle hinweg als Thin Client.

bereits beschrieben, eine konsistente Darstellung des Dialogzustands möglich ist. Wird im Fall der multimodalen Darstellung ein konsistentes Dialogmodell (DC) und eine konsistente abstrakte Darstellung (LP) über verschiedenen parallele Darstellungsmodalitäten aufrechterhalten (Änderungen in der einen Modalität werden über das Modell auf die anderen Modalitäten synchronisiert, dies wurde beispielsweise im EMBASSI Projekt umgesetzt, s. Abschnitt 7.1), so wird hier der Zustand der proximalen (anwendungsseitigen) und distalen (entfernten clientseitigen) konsistent gehalten.

Um die Zustände über eine Netzwerkverbindung konsistent zu halten bedarf es eines Protokolls, welches Benutzereingaben und den Zustand des Modells zwischen den Instanzen kommuniziert. Eine Analyse existierender Protokolle wird von Lanz [Lan05] beschrieben. Für eine detaillierte Beschreibung der verglichenen Ansätze und eine Spezifikation eines Protokolls für eine verteilte Umsetzung der Darstellungsplattform wird auf diese Arbeit verwiesen.

Die *browserbasierte Lösung* nutzt den W3C Standard XForms [DKMR03], auf welchen die abstrakten Schichten des DC und LP aufbauen. Die Bereitstellung als serverbasierte Anwendung kann so über eine Serverkomponente erfolgen, welche als eine Darstellungskomponente in das Framework eingebunden ist und XForms User Interfaces versendet und Instanzdaten empfängt. Der Transformationsaufwand ist hierbei gering, da das Framework bereits auf diesem Format aufbaut. Ob die Industrie in absehbarer Zeit XForms fähige Browser zur Verfügung stellen wird ist zwar nicht klar, jedoch werden von Drittanbietern bereits Plug-ins für den Microsoft Internet Explorer zur Verfügung gestellt, bzw. mit dem XSmiles Browser steht eine Open Source Implementierung in Java zur Verfügung. Eine stets aktuelle Übersicht verfügbarer Implementierungen findet sich auf der Website des W3C [Boy05].

All diesen Einsatzszenarien ist gemeinsam, dass die auf dieselben grundlegenden Prozessschritte, wie sie bereits im Entwicklungsmodell identifiziert und in der Systemarchitektur konkretisiert wurden. Im verbleibenden Teil dieses Abschnittes werden die einzelnen Komponenten detailliert beschrieben und diskutiert werden.

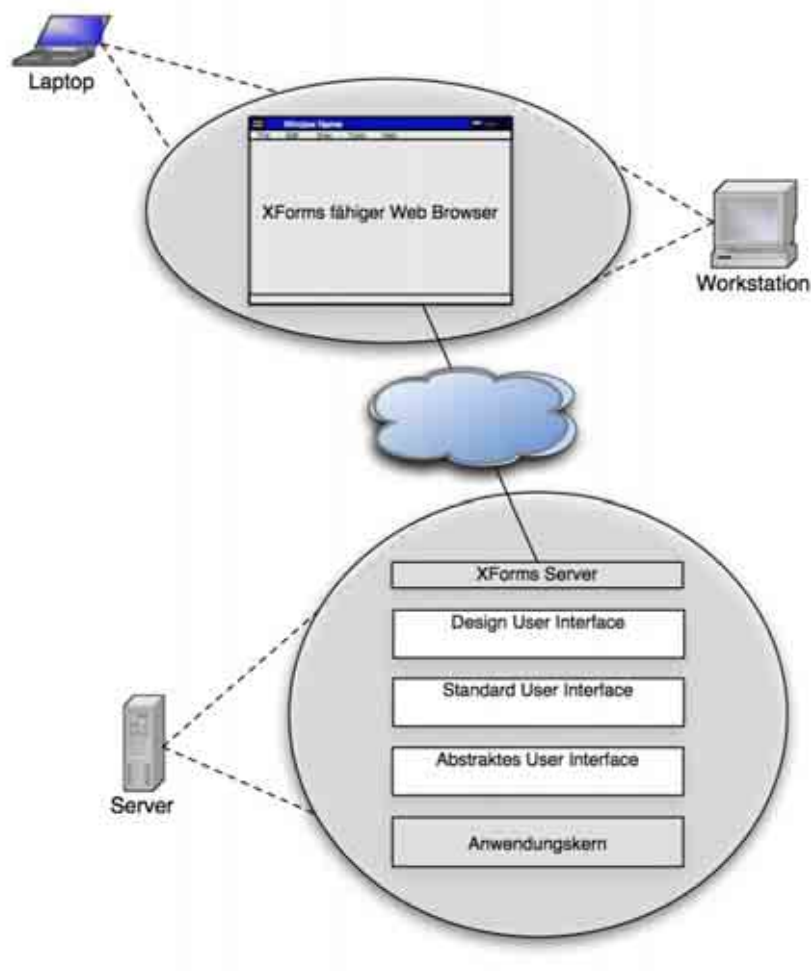


Abbildung 6.7: Einsatz über eine Netzwerkschnittstelle hinweg als Browser-basierte XFORMS Anwendung.

6.1.3 Anwendungsintegration

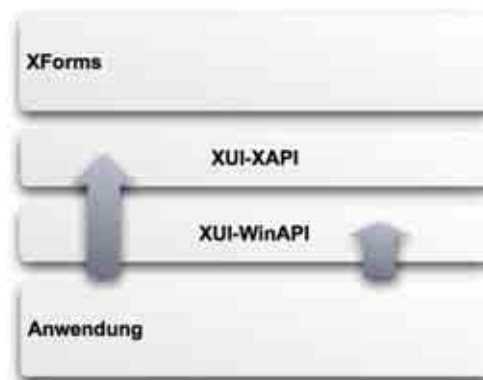


Abbildung 6.8: Anwendungsintegration über zwei Abstraktionsebenen.

Anwendungen werden in die Darstellungsplattform über eine API angebunden. Die systeminternen Prozesse werden somit weitestgehend vor dem Entwickler in Methodenaufrufen gekapselt und somit versteckt. Für den Entwickler verhält sich das System hierbei wie ein User Interface Toolkit. Die eigentliche Darstellung und Darstellungsanpassung erfolgt entweder für den Entwickler transparent oder dann durch Modifikation der Darstellung in der Entwicklungsumgebung. Der intern ablaufende mehrschrittige Darstellungsprozess wird gekapselt.

Der Entwickler kann zwischen zwei Programmierschnittstellen wählen, die sich in ihrer Abstraktionsebene unterscheiden. Die *Low-Level API* bietet den vollen Zugriff auf das Darstellungssystem. Die *High-Level API* baut auf die Low-Level API auf und kapselt die Implementierungsdetails der internen Datenstrukturen.

Die XAPI erlaubt einen direkten Zugriff auf die Datenstrukturen des Darstellungssystems, welches intern eine XFORMS Datenstruktur aufbaut. Es kann damit direkt auf XFORMS Modell und Elemente der Darstellung zugegriffen werden. Der Entwickler erhält somit volle Kontrolle über alle Aspekte der Oberfläche. Auf der anderen Seite erfordert diese Schnittstelle einige Einarbeitung, da sie zugleich wesentlich komplexer ist. Weiterhin unterscheidet sich das Objektmodell von dem üblicher User Interface Toolkits. Die XAPI umfasst ca. 69 Klassen und implementiert 100% der internen Datenstruktur.

Die WinXAPI ermöglicht den Zugriff auf das Darstellungssystem in derselben Form, wie dies in gängigen User Interface Toolkits der Fall ist. Ein Entwickler kann so weitestgehend die bestehenden Kenntnisse der Programmierung von Benutzerschnittstellen weiterverwenden ohne irgendwelche Besonderheiten bezüglich der plattformübergreifenden Anpassungen beachten zu müssen. Die Migration zwischen einem Toolkit und den Darstellungssystem fällt leicht. Der Umfang der verfügbaren Elemente umfasst ca 21 Objekte der WinForms API und implementiert nur 40% der internen Datenstrukturen.

Beide Programmierschnittstellen dienen verschiedenen Einsatzzwecken und adressieren verschiedene Aspekte der aus Kapitel 2 erarbeiteten Anforderungen. Die *High-Level API* dient in erster Linie der Reduzierung der *Komplexität*, und der Verbesserung der *Kompatibilität* mit existierenden Entwicklungsprozessen. Die *Low-Level API* adressiert hingegen vielmehr die Umsetzung der funktionalen Ziele, die die Mächtigkeit des XFORMS/XUI-Konzeptes ausmachen. Durch die volle Ausschöpfung der XFORMS-Spezifikation wird hier insbesondere der Aspekt der *Plattformunabhängigkeit* und der *Skalierbarkeit* unterstützt.

Abbildung 6.9 gibt eine komponentenbasierte Sicht der Anwendungsintegration wieder. Wie hier dargestellt, kapselt die Anwendungsintegration das Modell und die Darstellung über definierte Programmierschnittstellen. Die hier dargestellten Schnittstellen unterscheiden sich im wesentlichen durch ihren Umfang und durch die Nähe zur tatsächlichen Implementierung des XFORMS Kerns.

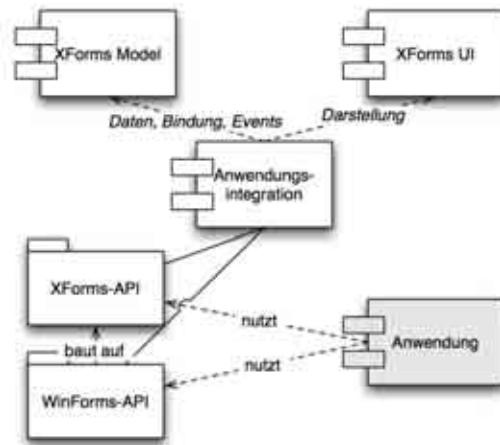


Abbildung 6.9: Komponenten der Programmierschnittstellen.

Die mit *XAPI* bezeichnete Schnittstelle unterstützt die wesentlichen Aspekte von XFORMS und kennt die Unterscheidung in *Modell* und *Präsentation* sowie die *Bindung* von Elementen an Daten, und das Ereignismodell in dem *Benutzeraktionen* beschrieben und an Ereignisse gebunden werden können.

Die näher an die Anwendung orientierte *WinXAPI* Schnittstelle versteckt diese Aspekte, um so eine Implementierung in gewohnter Weise ermöglicht. Sie bietet dem Programmierer nur die Elemente an, welche ihm aus User Interface Toolkits (hier den *Windows Forms*) bekannt sind. Aus diesem Grund ist der Funktionsumfang geringer als bei der XAPI. Die Komponenten zur Verwaltung der Instanzdaten oder Bindung an diese ebenso wie die Aktionen werden nicht benötigt da diese für den Entwickler transparent automatisch erzeugt werden. Einen Auszug aus dem Klassendiagramm der *WinXAPI* zeigt die Abbildung 6.10.

Die Klasse *WinXAPIUIElement* stellt die abstrakte Oberklasse für alle Oberflächenelemente der *WinXAPI* dar. Sie erzeugt, analog zur XAPI, einen eindeutigen Elementbezeichner, welcher für das abstrakte Modell benötigt wird. Dieser kann über das Attribut *Name* abgefragt werden. Darüber hinaus definiert sie Attribute und Methoden, welche für alle Oberflächenelemente einheitlich sind. Dazu gehören Elemente der Ereignisverarbeitung sowie der Darstellung.

Die Klasse *WinXAPIBoundUIElement* dient als abstrakte Definition für die Oberflächenelemente der *WinXAPI*, welche an Knoten in den Instanzdaten gebunden werden müssen. Im Gegensatz zur XAPI werden hier dem Anwender allerdings keine zusätzlichen Attribute zur Bindung von Bedienelementen an Instanzknoten zur Verfügung gestellt, dies wird intern geregelt. Dafür definiert sie private Attribute zum Zugriff auf ein Objekt der abstrakten Schicht, welches an Datenknoten gebunden werden kann. So kennt etwa jedes Element der *WinXAPI* sein Gegenstück im abstrakten Modell (durch das Attribut *XUIElement*). Wird für ein Oberflächenelement der *WinXAPI* der Wert durch den Entwickler abgefragt oder gesetzt, so greift dieses über sein abstraktes Gegenstück auf die Instanzdaten der Applikation zu.

Von der Klasse *WinXAPIBoundUIElement* abgeleitet sind weitere Klassen, welche diese weiter spezialisieren. Analog zur XFORMS-nahen API sind das *WinXAPIUIContainerElement* für Elemente, welche andere Objekte aufnehmen können, sowie *WinXAPIAbstractSelect* und *WinXAPIAbstractSelectBoxes* für Oberflächenobjekte, die Auswahlmöglichkeiten repräsentieren.

Der Aufbau der *Low-Level API* orientiert sich an der Hierarchie des abstrakten Modells. Dabei wird grob in die Bereiche der *Model*-, *Action*-, *Binding*- und *UI*-Elemente unterteilt. Alle Komponenten sind von einer abstrakten Oberklasse abgeleitet, welche sicherstellt, dass definierten Elementen ein eindeutiger Elementbezeichner zugewiesen wird. Dieser ist für die korrekte Verwaltung der Objekte im abstrakten Modell Voraussetzung und kann von jeder Stelle in der Programmierschnitt-

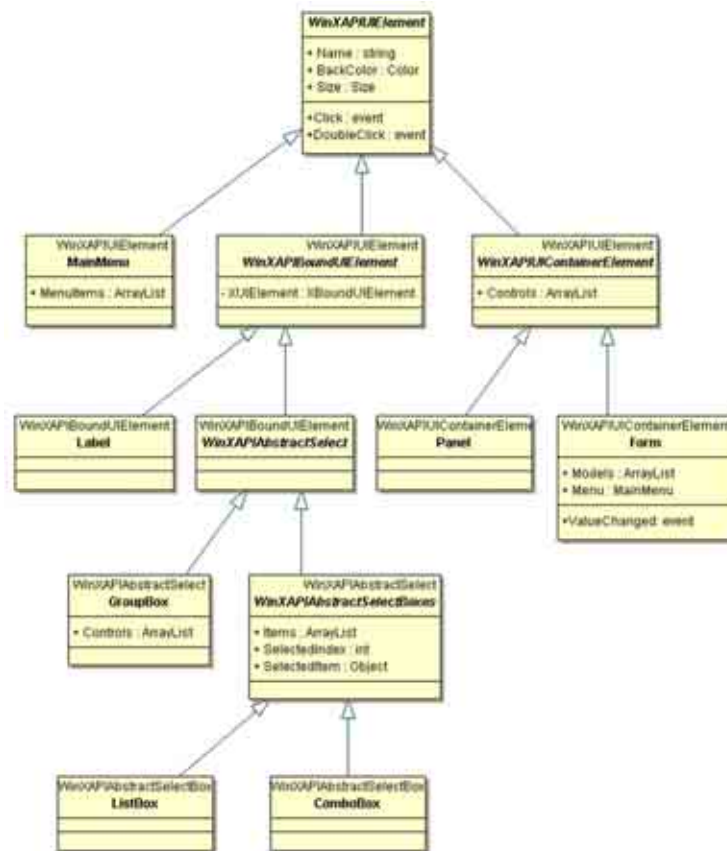


Abbildung 6.10: Auszug aus der Klassenhierarchie der WinXAPI.

stelle abgefragt werden. Abbildung 6.11 stellt die Aufteilung in die unterschiedlichen Komponenten dar.

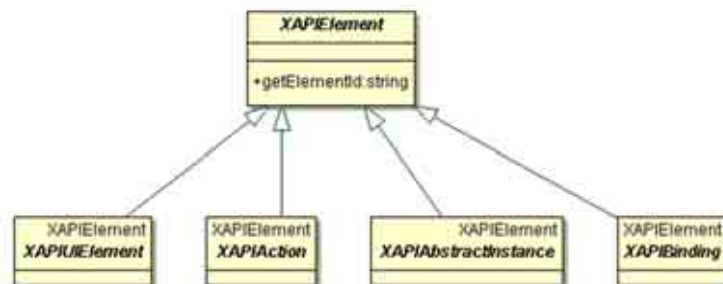


Abbildung 6.11: Die Aufteilung der an XFORMS orientierten Programmierschnittstelle in ihre Hauptkomponenten

Hierbei stellen die Klassen `XAPIUIElement`, `XAPIAction`, `XAPIAbstractInstance` und `XAPIBinding` ihrerseits abstrakte Spezifikationen für die Bereiche der Oberflächenelemente, Ereignisse, Instanzdaten und die Bindung an die Instanzdaten dar.

Die folgenden zwei Auszüge aus Programmen zeigen Beispiele für die Verwendung beider Programmierschnittstellen. Das erste Beispiel 6.1 zeigt die Verwendung der WinXAPI zur Erzeugung eines einfachen Dialoges mit einer Schaltfläche, die eine Aktion auslösen kann. Die Umsetzung ist nahezu identisch mit der entsprechenden Implementierung mit *Microsoft Windows Forms*. Das bedeutet, dass durch Referenzierung der entsprechenden Pakete (*Assemblies*) eine Integration auf diese Weise sehr schnell und unkompliziert zu realisieren ist.

Listing 6.1: Verwendung der Integrationskomponenten der High-Level (WinXAPI) API

```

1 class Beispiel
2 {
3     // Die Main Methode
4     static void Main(string[] args)
5     {
6         // Neues Formfenster anlegen
7         Form form = new Form();
8
9         // Titel des Formfensters setzen
10        form.Text = "HelloWorld";
11
12        // Erstellen einer Schaltfläche
13        Button bReset = new Button();
14        bReset.Text = "Reset";
15
16        // Ein Eventhandler wird registriert
17        bReset.Click += new EventHandler(bReset_Click);
18
19        // Tooltip für Button anlegen
20        ToolTip ttButton = new ToolTip();
21        ttButton.SetToolTip(bReset, "Daten zurücksetzen");
22
23        // Element wird dem Form hinzugefügt
24        form.Controls.Add(bReset);
25
26        // Starten der Applikation
27        form.Run();
28    }
29
30    // Die Methode zur Behandlung der Events
31    private void bReset_Click(object sender, EventArgs e)
32    {
33        // Aktionen
34    }
35 }

```

Der zweite Auszug 6.2 zeigt die entsprechende Umsetzung einer einfachen Anwendung mit einer Texteingabe, welche die Referenz auf ein Datenobjekt übergeben bekommt, um diese anzuzeigen und deren Veränderung zuzulassen. Die Referenz wird hierbei automatisch auf ein intern aus der Instanz der Objektes erzeugtes XML-Dokumentenobjekt angelegt. Auch hier erfolgt also eine Kapselung der internen XML-Verarbeitung durch objektorientierte Schnittstellen.

Listing 6.2: Verwendung der Integrationskomponenten der XAPI.

```

1 class Beispiel
2 {
3     static void Main(string[] args)
4     {
5         // Formfenster anlegen
6         Form form = new Form();
7
8         // Modellelement anlegen
9         XAPIModel model = new XAPIModel();
10        // Modellelement dem Formfenster hinzufügen
11        form.Models.Add(model);
12
13        // Integration der Daten durch Referenz auf Instanz
14        Person p = new Person();
15        XAPIInstance instance = new XAPIInstance(p);
16        model.Instances.Add(instance);
17
18        // TextBox zur Namenseingabe
19        TextBox tb = new TextBox("Name:");
20        tb.Ref = new XAPIRef(instance, "Name");
21        form.Controls.Add(tb);
22
23        // Starten der Applikation
24        form.Run();
25    }
26 }
27
28 class Person
29 {
30     public string Name = "Mustermann";
31     public string Vorname = "Max";
32 }

```

Die Konzeption der Anwendungsschnittstelle sowie deren Umsetzung einer Beispielimplementierung in Microsoft C# auf der .Net Plattform wird von Herold ausführlich in [Her05] beschrieben. Aus diesem Grund wird an dieser Stelle nur exemplarisch auf den Einsatz der API eingegangen.

Wie gezeigt wurde, können über die beiden Schnittstellen so verschiedene Abstraktionsebenen realisiert und verschiedenen Ebenen der Entwickler-Expertise angesprochen werden. Neben der üblichen Entwicklungsweise stehen auf diese Weise auch die Möglichkeiten von XFORMS zur Verfügung, Daten und Präsentation in hohem Maße zu separieren und direkt in XML-basierte Datenstrukturen zu schreiben.

6.1.4 Datenmodell

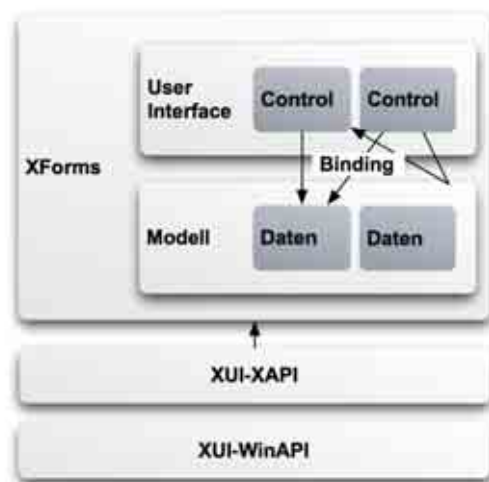


Abbildung 6.12: Die Daten werden innerhalb des Modells verwaltet.

Das Modell hat die Aufgabe die Daten, welche über die Benutzerschnittstelle zugänglich gemacht werden, zu verwalten.

Entsprechend der XFORMS Architektur (s. Abschnitt 4.3.3.1) werden die Daten getrennt von der Darstellung in dem Modell gehalten. Daten können so mit verschiedenen Darstellungen kombiniert und bearbeitet werden. Die Darstellung sowie die Bearbeitung in den Elementen referenziert stets auf die Elemente des Modells, so dass die Integrität der Daten sichergestellt ist. Dieses Verfahren eignet sich daher auch für die Darstellung in multimodalen Systemen, in denen die Integration der Ein- und Ausgaben einen wichtigen Aspekt darstellt.

Weiterhin können in einem Modell Abhängigkeiten von Daten untereinander, sowie Aktionen und Berechnungsprozeduren beschrieben werden. Die Verknüpfung der Darstellung und der Daten erfolgt über Bindungen (*Bindings*) die mittels XPATH-Ausdrücken [CD99] innerhalb der Benutzerschnittstelle, aber auch im Modell definiert werden können. Die Umsetzung des Modells in diesem System orientiert sich unmittelbar an der XFORMS-Spezifikation [Dub03, DKMR03].

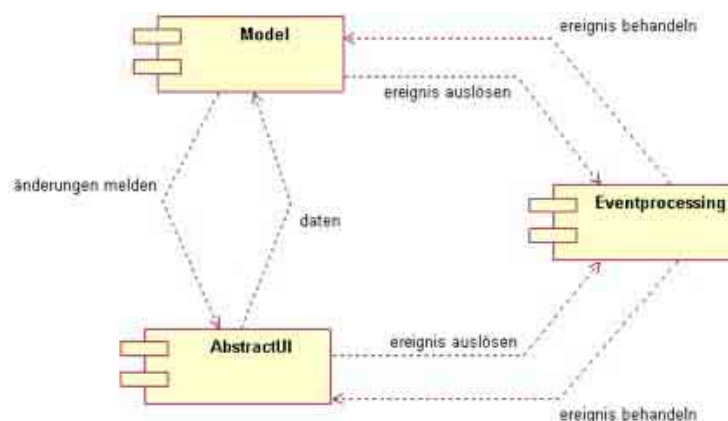


Abbildung 6.13: Rolle des Modells in Zusammenarbeit mit der abstrakten Darstellung.

Abbildung 6.13 zeigt die Einbettung des Modells in die Komponentensicht des Systems. Das Modell wirkt als Verwaltungseinheit für die Daten einer Benutzerschnittstelle. Ein Modell kann verschiedene Datenelemente enthalten, auf welche die Elemente der Benutzerschnittstelle aber auch

die Anwendung selbst zugreifen kann. Der Inhalt des Modells wird allen an dieses Modell gebundenen Elemente gemeldet und die Darstellung synchronisiert. Unter Verwendung des Ereignismodells können auch andere Elemente Änderungen abfangen. Das Modell hat in der Architektur des Darstellungssystems also folgende die Aufgabe, die Konsistenz der Daten in Unabhängigkeit der Darstellung zu gewährleisten, die Daten an die Oberfläche und an die Schnittstelle zu Anwendung (WinXAPI / XAPI) bereitzustellen, und Methoden zur datenorientierten Ereignisbehandlung und Berechnungsprozeduren zu realisieren.

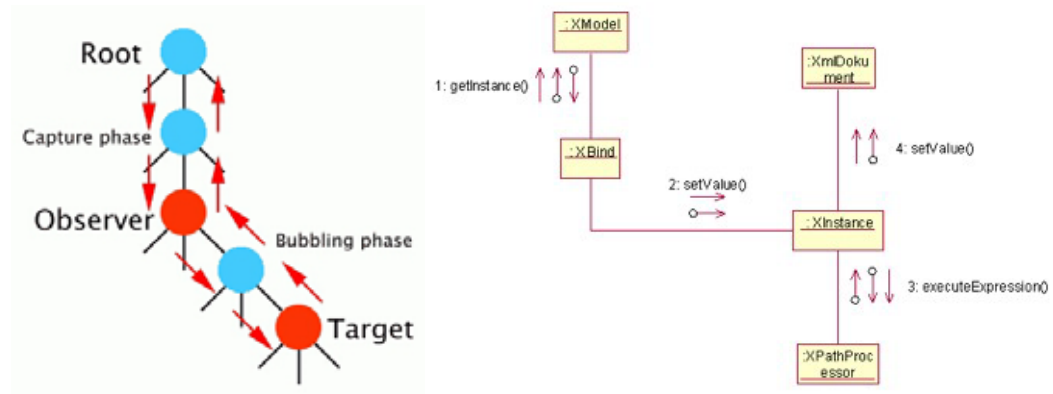


Abbildung 6.14: Beispiel für die Verarbeitung von XML Ereignissen.

Abbildung 6.14 zeigt den Prozess der Ereignisverarbeitung innerhalb des Modells. Die Ereignisse innerhalb der abstrakten Ebene der Benutzerschnittstelle werden ebenfalls konform zu der XFORMS Spezifikation über die *XML Events* Schnittstelle [MPR03, Pix00] abgehandelt. Dieses XML-spezifische Vorgehen der Ereignisverarbeitung wurde auf Kompatibilitätsgründen angewandt.

Abbildung 6.14 zeigt weiterhin, dass Veränderungen in den Daten, sei es durch die Anwendung oder durch den Benutzer, stets einen Prozess der Synchronisierung in Gang setzen. Dieser greift über die Instanzdaten-Referenz, und der darin mittels XPATH-Ausdrücken beschriebenen Verweise (*Bindungen*) auf die Datenfelder des Modells zugreift. Synchron dazu werden in den Daten der Benutzerschnittstelle die entsprechenden Inhalte der Darstellung aktualisiert.

Wie gezeigt wurde, konnte das Konzept, welches in XFORMS spezifiziert wurde, in der Darstellungsplattform für plattformübergreifende Benutzerschnittstellen als integrierende Datenkomponente eingebracht werden. Diese steht als Vermittler zwischen Anwendung und Darstellung zur Verfügung und abstrahiert die Daten von deren konkreter Darstellung. Die Konzeption und Beispielimplementierung wurde von Kopp in [Kop04] beschrieben.

6.1.5 Abstrakte Präsentation

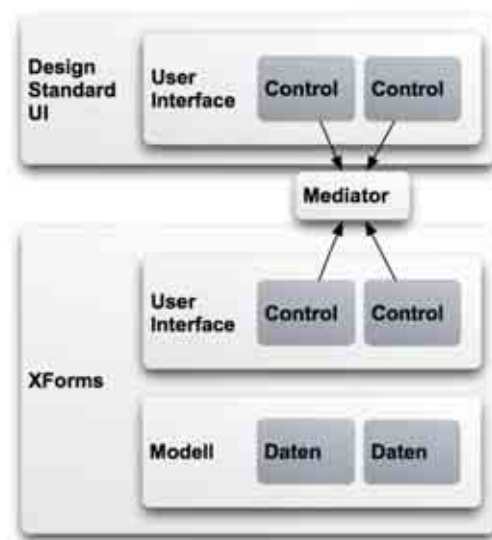


Abbildung 6.15: Die abstrakte Präsentation

Die abstrakte Präsentation stellt die plattform- und geräteunabhängige Repräsentation der Elemente der Oberfläche dar. Sie umfasst eine abstrakte Spezifikation der Elemente deren statische Inhalte und Stile. Diese Ebene baut hierbei auf der XFORMS Benutzerschnittstelle [Dub03, DKMR03] auf, welches als zweiter Bestandteil der XFORMS Architektur unabhängig von dem Datenmodell definiert, und über Referenzen an dieses gebunden werden kann (s. Abschnitt 6.1.4).

Diese Schicht setzt in erster Linie die *Form Controls* [DKMR03, Abschnitt 8] um und erweitert diese um die Elemente der Menüleiste *XMenu* und einem Gruppierungselement *XUI*. Wenngleich XFORMS nicht festlegt, wie eine Benutzerschnittstelle zur Darstellung gebracht werden soll und daher die Verwendung verschiedener Beschreibungssprachen vorsieht, sollen hier die allgemeinen Elemente genutzt werden. Die XFORMS Spezifikation beschreibt einen Satz von ca. 20 abstrakten Elementen. Diese Elemente sollen in erster Linie die plattform- und darstellungsunabhängige Beschreibung von Elementen ermöglichen und erscheinen daher für das vorliegende System als geeignet.

Die Oberflächen-Elemente werden in der XFORMS Spezifikation in abstrakter Weise primär funktional beschrieben. Diese Elemente lassen sich daher in vielen Fällen nicht direkt in ein Element gängiger User Interface Toolkits übersetzen, sondern setzt sich aus einer Gruppe von Einzelelementen zusammen.

Abbildung 6.16 zeigt, wie die Hinweis-Texte als *Tooltip* und eine Fehlermeldung als zusätzlich eingeblendeter Text dargestellt werden, obwohl beide Texte Bestandteile eines XFORMS Oberflächen-elements sind. Ein Element der XForms-Spezifikation entspricht in der Regel daher einem kombinierten Element gängiger Toolkits. Dies hat Implikationen für die Darstellung einzelner Elemente in der Oberfläche, da hierdurch das Layout komplexer wird. Aspekte der Zugänglichkeit von Benutzerschnittstellen, wie Navigationsindices, Gruppierungen, Beschreibungs- und Hilfetexte sind direkt in die Elemente hinein definiert.

Der Vorteil dieser Abstraktion, die über deutlich mehr obligatorische Informationen zu den Elementen verfügen als gängige Toolkit Elemente, besteht darin, dass so ausreichend Informationen für alternative Darstellungsweisen vorhanden sind. Visuelle Hinweise, die ansonsten zur Orientierung in einer Darstellung vorhanden sind, gehen bei Änderungen des Layouts, oder bei der Darstellung in anderen Modalitäten meist verloren. Die direkte Integration beschreibender Texte und Navigationspositionen macht die Informationen für die Darstellung verfügbar.



Abbildung 6.16: Darstellung eines abstrakten XFORMS Elements

Der Nachteil der Abstraktion besteht in erster Linie in der größeren Komplexität der einzelnen Elemente. Das Layout für diese kombinierten Elemente stellen für der Darstellung mittels eines Toolkits ein Problem dar, das es nicht spezifiziert werden kann. XForms betrachtet diese als ein Element und gibt keine Vorschläge wie die konkrete Gestaltung erfolgen soll. Eine geringe Handhabe erhält der Entwickler über das Komplexitätsattribut, welches meist die Komplexität der Gestaltung (z. B. bei einem Datumsfeld) regeln lässt. Mehr wird hierzu im nächsten Abschnitt 6.1.6 diskutiert werden.

Da XFORMS dafür konzipiert wurde, in einer *Host Language* eingebettet zu werden, wurde ein Dokumenttyp definiert, welcher mit dem Basis-Element `XUI` als Container für diese erweiterte XForms User Interfaces genutzt werden kann. Diese Dokumenttyp wird im Rahmen des Systems als *extendible User Interface Language (XUI)* bezeichnet.

Die Erweiterungen des XFORMS Standards wurden auf einen minimalen Satz an Elementen beschränkt. Die Erweiterung ist so gestaltet, dass die Einbettung in andere *Host Languages* weiterhin möglich ist.

Abbildung 6.3 zeigt einen Ausschnitt eines XUI Dokumentes. Innerhalb des Dokumentes werden das Modell (`model`) und die Benutzerschnittstelle (`form`) beschrieben. Die Verknüpfung erfolgt über die Bindung (`bind`).

Das Beispiel zeigt das Menü-Element (`xui:menu`) sowie Ausgabe-Elemente (`output`), die meist als Text dargestellt werden, einen Aktions-Element (`trigger`), der einer Schaltfläche entspricht und einem Auswahlelement (`select1`), das in verschiedenen Darstellungsweisen entweder als Radiobuttons (`full`), Liste (`compact`) oder Combobox (`minimal`) visualisiert wird (s. Abbildung 6.17).

Listing 6.3: Beispiel für die kanonische Form eines XUI Dokumentes.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/" href="style.css"?>
3 <xui:xui xmlns:xui="http://www.zgdv.de/xui"
4       xmlns:ev="http://www.w3.org/2001/xml-events"
5       xmlns="http://www.w3.org/2002/xforms">
6   <model xui:id="modell">
7     <instance>
8       <data>
9         <!-- data -->
10      </data>
11    </instance>
12    <bind xui:id="total" nodeset="total" calculate="data/itemsum_+_data/tax"/>
13  </model>
14  <xui:form>
15    <xui:menu>
16      <!-- xui:menuItem -->
17    </xui:menu>
18    <label>XUI - Shoppingcard </label>
19    <output>
20      <label>Artikel </label>
21    </output>
22    <input bind="amount1" incremental="true">
23      <label ref=".." />
24      <alert>Bitte geben sie eine Zahl ein.</alert>
25    </input>
26    <trigger xui:id="okButton">
27      <label>Bestellen >></label>
28      <toggle ev:event="DOMActivate" case="out"/>

```

```

29         </trigger>
30         <select1 bind="versand" appearance="full">
31             <label>Versand</label>
32             <item>
33                 <label>Abholung</label>
34                 <value>0</value>
35             </item>
36             <!-- items -->
37         </select1>
38     </xui:form>
39 </xui:xui>

```

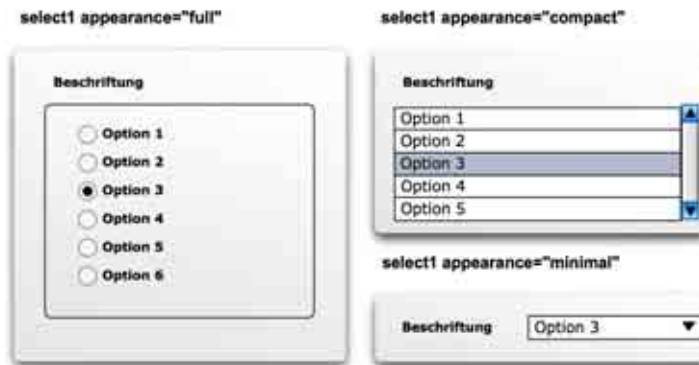


Abbildung 6.17: Auswirkung der Appearance Eigenschaft auf die Darstellung

Gemäß der in der Abschnitt 2.4 identifizierten Anforderungen an den Entwicklungsprozess, wurde bei der Konzeption des Systems auf die Verwendung von Standards wie XML [Mar01], XFORMS [DKMR03], *Cascading Stylesheets (CSS)* [LB99b, LB99a] geachtet. Die im Rahmen der XUI-Spezifikation erfolgte Erweiterung hat zum Ziel, eine weitestgehende Konformität mit diesen Standards zu erreichen.

Die Beschreibung der Darstellung erfolgt in der Ebene der abstrakten Präsentation entweder *konkret* oder *abstrakt*.

Die *abstrakte Darstellung* sieht vor, dass Farben, typographische Informationen, usw. auf der Ebene von semantischen Beschreibungen wie beispielsweise *Wichtig*, *Fehler*, *Frage* oder ähnlich beschreiben werden. Die Design Standards für diese Beschreibungen werden dann angewendet. Dies entspricht weitestgehend dem Prinzip von Systemfarbpaletten wie sie bei der Gestaltung von plattformkonsistenten Anwendungen empfohlen werden (s. Abbildung 6.18).

Die Positionierung kann ebenfalls unspezifiziert bleiben. Das Layout erfolgt dann anhand des Navigationsindex oder auf Basis der hierarchischen Position oder der Einbettung in eine Gruppe.

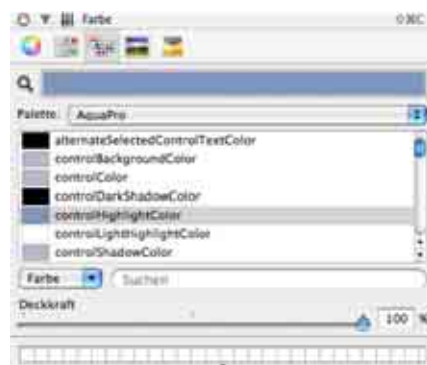


Abbildung 6.18: Beispiel für eine Systempalette zur konsistenten Formatierung.

Die *konkrete Darstellung* sieht vor, dass Werte für Farben und Position explizit gesetzt werden. Zur Spezifizierung von Formatierungen wird auf die *Cascading Stylesheets (CSS)* [LB99b, LB99a]

zurückgegriffen. Dies bedeutet weiterhin, dass Formatierungen entweder innerhalb der Elemente, oder extern in *Stylesheets* beschrieben werden können.

Diese Vorgehensweise ist insbesondere bei der Verfolgung der Strategien der hierarchischen Entwicklungspfade (s. Abschnitt 5.2.2) wichtig. Dann wird die Darstellung in dieser Ebene bereits über die API auf die Anforderungen der primären Plattform optimiert und erst von der Design Standards und durch die Modifikation durch den Designer angepasst. Kaskadierende Formatierungen für die abstrakte Beschreibung können dann von den konkreten Formatierungen für konkrete Präsentationen überschrieben werden.

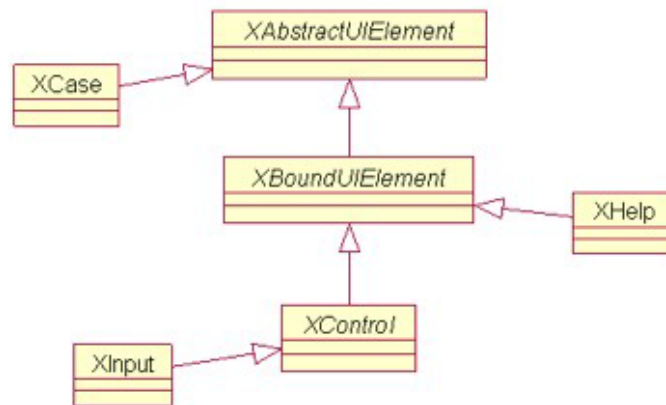


Abbildung 6.19: Vererbungshierarchie der abstrakten Darstellungselemente.

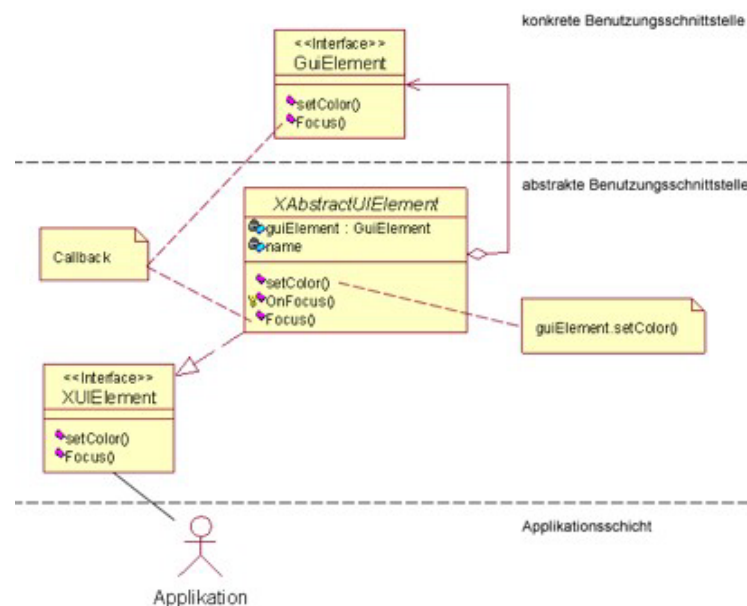


Abbildung 6.20: Zugriff auf die konkreten Darstellungselemente über das abstrakte User Interface.

Die abstrakte Benutzerschnittstelle dient als Zwischenschicht zwischen konkreter Darstellung und der Anwendung, bzw. der Programmierschnittstelle. Diese Abstraktion ist notwendig um verschiedene Einsatzszenarien (s. Abschnitt 6.1.2) zu realisieren und die Bindung zwischen Datenmodell und Benutzerschnittstelle zu ermöglichen. Dadurch werden Daten und Interaktion zwar aneinander, aber nicht an eine konkrete Darstellung gebunden.

Dieser Teil des System ist identisch für alle Plattformen auf denen das System eingesetzt wird. Die Koppelung an die plattformspezifische Darstellung erfolgt über die konkreten Elemente, die als

Ableitungen des `GuiElement`-Interfaces als Referenz übergeben werden (s. Abbildung 6.20). Die Verknüpfung erfolgt über eine Brücke, welche das Vermittler (*Mediator*) Entwurfsmuster realisiert. Dieser Mediator kontrolliert hierbei den Nachrichtenfluss in beide Richtungen und unterscheidet Benutzerein- und Systemausgabe.

6.1.6 Standard Präsentation

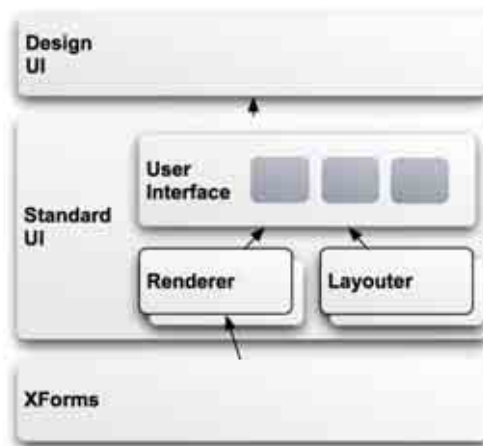


Abbildung 6.21: Die Komponenten der Standard Präsentation

Die Standard Darstellung ist jene Repräsentation der Benutzerschnittstelle, welche als Resultat des ersten plattformspezifischen Darstellungsprozesse erzeugt wurde. Sie erfolgt anhand einheitlicher, fest vorgegebener Übersetzungsschritte. Abbildung 6.21 zeigt, wie die Komponenten des Darstellungssystems auf der Ebene der Standard Präsentation zusammenhängen.

Die Standard Darstellung ist der erste Schritt der konkreten Darstellung. In diesem werden Instanzen konkreter Toolkit Elemente der spezifischen Plattform erzeugt.

Die Initialisierung des Dialoges führt dazu, dass aus der Laufzeitumgebung die aktuelle Plattform ausgelesen wird. Derzeit erfolgt keine feinere Profilierung der Geräte. Lediglich die `Environment.OSVersion.Platform` Variable gibt darüber Auskunft welche Plattform aktuell genutzt wird.

Entsprechend der aktuellen Plattform wird die abstrakte Darstellung nun einem *Render*-Modul übergeben, welches die `GuiElement` Instanzen für die entsprechende Plattform erzeugt und mit dem Datenmodell verknüpft, bzw. die Formatierungen aus dem Stylesheet anwendet. Ein zweiter Prozess, der *Layouter* sorgt dann dafür, dass die Elemente entsprechend der Anforderungen der aktuellen Plattform angeordnet werden.

Das realisierte System verfügt über drei *Render / Layout* Komponenten, welche die drei Plattformen unterstützen.



Abbildung 6.22: Beispielhafte Darstellung einer einfachen Anwendung auf drei Plattformen.

Abbildung 6.22 zeigt eine einfache Anwendung auf den drei Plattformen, für die bereits Render Module existieren. Während die Darstellung auf dem PC recht einfach über die Positionierung der Elemente erfolgen kann (diese Plattform wird entsprechend der Annahme der hierarchischen Entwicklungspfade als primäre Plattform angenommen), muss die Darstellung für die *Smart-Device*

Plattformen daraus abgeleitet werden. Aufgrund der räumlichen Einschränkung ist hier auch größerer Anpassungsaufwand notwendig.

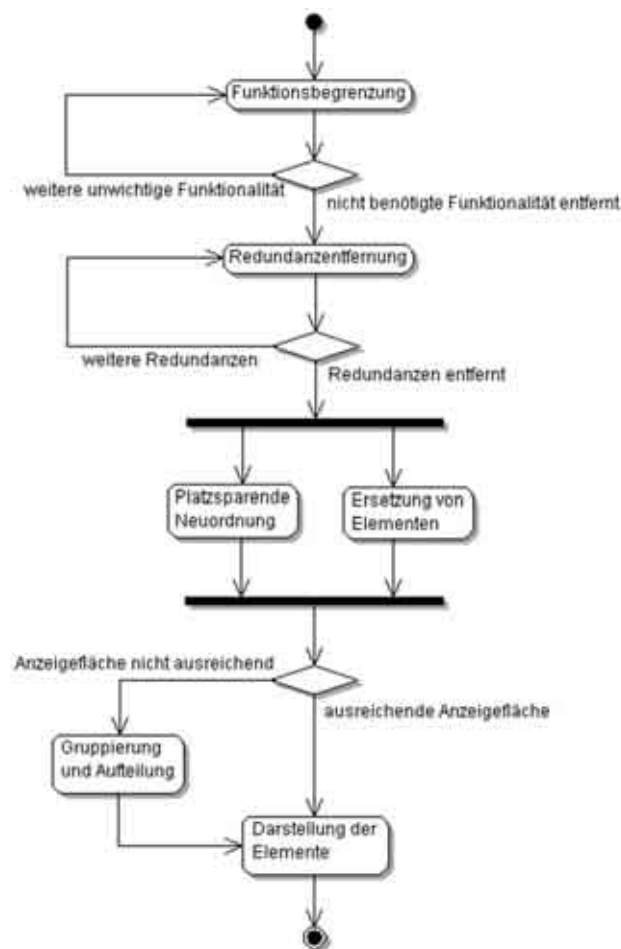


Abbildung 6.23: Beispielhafte Darstellung eines Reduktionsprozesses.

Abbildung 6.23 zeigt ein Beispiel für eine Pipeline, wie sie in einem Layout-Prozess angewendet werden kann. Das hier umgesetzte System fokussiert hierbei die Prozesse der *platzsparenden Neuordnung*, der *Elementersetzung* und der *Gruppierung und Aufteilung* (s. Abschnitt 5.3).

Zur Beschreibung der konkreten Darstellung wird auf dieselben Stil-Elemente zugegriffen, die über die API in der abstrakten Darstellung definiert wurden. Für jede Darstellungsplattform wird auf dieser Ebene ein neuer Satz an Stil-Attributen angelegt. Elementersetzungen werden über die das *Appearance*-Attribut beschrieben, Positionierungen und Neuordnungen durch die Anpassung der Positioneigenschaften des Stil-Elements, Anordnungen durch die Erzeugung von Dialogen.

Die Umsetzung der Darstellung verschiedener Modalitäten wurde im Rahmen des in Abschnitt 7.1 beschriebenen Forschungsprojekts umgesetzt und validiert. Das hier vorgestellte System ist in wesentlichen Bestandteilen aus dem dort entwickelten System abgeleitet und entsprechend der Anforderungen an den Entwicklungsprozess in Abschnitt 2.4 weiterentwickelt worden. Eine wesentliche Anpassung war hierbei der Verzicht auf die aufwändige Agenten-Architektur und Kommunikation. Die zugrundeliegende Umsetzung der XFORMS Benutzerschnittstelle und die Entscheidung der Entwicklung auf Basis einer vereinfachten API wurden im Rahmen dieses Projektes entwickelt und weitergeführt.

6.1.7 Design Präsentation

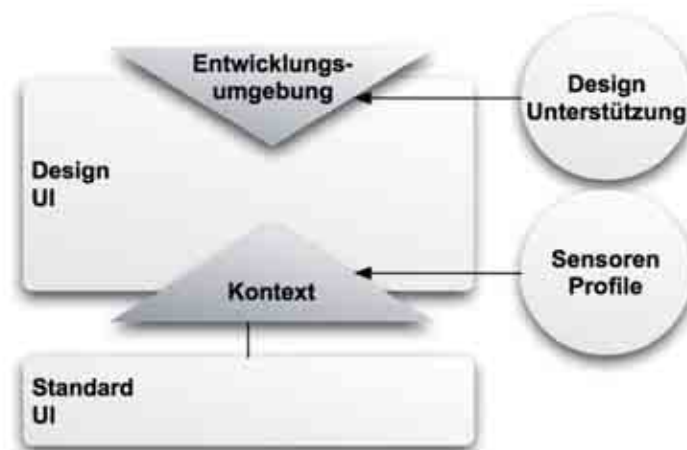


Abbildung 6.24: Die Design Präsentation.

Die Design Präsentation ist das Endprodukt der Anpassung und damit die endgültige Schnittstelle mit der der Benutzer konfrontiert wird. In dem letzten Anpassungsschritt fließen zum einen jene Optimierungen ein, welche der Designer, speziell für eine Plattform eingebracht hat, zum anderen werden hier im Darstellungsprozess die Kontextparameter berücksichtigt, die zur Optimierung der Darstellung auf die Benutzungssituation notwendig sind.

Die *Anpassung durch den Designer* erfolgt in dem System in erster Linie in graphischer Form. Das bedeutet dass die Position, bzw. Gestaltung der Oberflächenelemente durch den Designer nochmals verändert werden kann. Die ist insbesondere vor dem Hintergrund notwendig, dass einer der wichtigsten Kritikpunkte an UIMS die schlechte Qualität der generierten Darstellung ist (s. Abschnitt 2.4). Die Möglichkeit der manuellen Anpassung wurde aus diesem Grund auch in die Anforderungen aufgenommen.

Die durch den Designer vorgenommenen Anpassungen werden im Entwicklungsprozess in gesonderte *Stylesheets* geschrieben. Diese Änderungen überlagern dann die Formatierungen, die aus den Standard-Design hervorgegangen sind. Ein Anwenden der Änderungen auf die allgemeine Darstellung kann ebenfalls erfolgen wie in Abschnitt 5.2.2 beschrieben. Die Umsetzung solcher graphischer Anpassungsmethoden wurde zugunsten der Entwicklung von Unterstützungsmethoden nicht vorgesehen. Die konzeptionellen Grundlagen für solche Arbeiten kann jedoch in Abschnitt 5 gefunden werden.

Bei der Gestaltung einer plattformoptimierten Darstellung sollte der Designer unter anderem dahingehend unterstützt werden, dass die Anpassungen nicht zu sehr zu Lasten der Konsistenz der Darstellung gehen. Methoden, die hierfür geeignet sind werden auf Basis der Transformationshypothese in Abschnitt 6.2 besprochen.

Neben der Designanpassung erfolgt auf der Ebene der Design Präsentation eine Anpassung der Darstellung an *Kontextparameter*. Die Verortung der Anpassung an dieser Stelle ist jedoch nicht zwingend und kann ebenfalls auf den Ebenen der abstrakten bzw. Standard Präsentation erfolgen. Dieser Ansatz wird insbesondere bei modellbasierten UIMS vorgesehen.

Ein zentraler Aspekt kontext-sensitiver Systeme ist neben der Erfassung und Operationalisierung des Kontextes vor allem die Frage, *wie* das System auf die Kontextparameter angepasst werden soll. Systeme wie MANNA [EVP01] setzen verschiedene Platfformeigenschaften in Präsentationsmodelle um, in PALIO [SPZ04] werden die Oberflächen anhand der Informationen über den Ort und Benutzer angepasst und Grundy und Zou vergeben Benutzerrollen um verschiedene Sichten auf Daten zu generieren [GH00]. In der EMBASSI-Plattform [Ric02a] wird das Wissen über verschiedene

Plattformeigenschaften durch Darstellungsmodule umgesetzt und behinderungsspezifische Benutzeranforderungen in der Konfiguration von Ein- und Ausgabegeräten umgesetzt.

Dey und Abowd [DA04] schlagen folgende Kategorien von kontext-spezifischen Anpassungsfunktionen vor:

Präsentation von Informationen und Diensten Anzeige von Kontextinformationen oder kontext-sensitive Auswahl von Diensten oder Anwendungen, z. B. Anzeige der Position in einer Karte.

Ausführung von Informationen und Diensten Ein Dienst wird automatisch ausgeführt, wenn eine bestimmte Konfiguration von Kontextparametern erfüllt ist, z.B. Mobiltelefon wird lautlos gestellt sobald man in das Büro kommt.

Annotation von Informationen mit Kontextdaten Informationen werden im Kontext abgelegt oder mit Kontextdaten versehen, z.B. Hinterlassen einer Nachricht an einem Ort oder Annotation von Informationen mit Orten an einer Baustelle [GTB02, BGT05].

Als weiterer Punkt soll hier noch die Anpassung von Diensten oder der Informationsanzeige eingeführt werden, da dies im Sinne der plattformübergreifenden Dienste von Relevanz erscheint. Notwendig ist hierbei die Anpassung von Informationen und Diensten: Dienste und deren Ansichten werden an den Benutzer, Plattform oder Kontext angepasst, z.B. Skalierung und Filterung von Oberflächen bei Benutzung mit Mobiltelefon.

Welche Konsequenz hat die Kontext-Adaption nun auf die Entwicklung von kontext-abhängigen Systemen? Je nach Anzahl der berücksichtigten Kontextparameter kann das System zu einer beliebig umfangreichen Komplexität anwachsen, mit jedem Parameter wächst der Raum der möglichen System-Ausprägungen um das Produkt der Abstufungen dieses Parameters ($n \times m \times o$ Problem). Ist die Anzahl der Abstufungen, bzw. die Spanne des Gültigkeitsraumes nicht klar definiert, lassen sich die Variationen nicht mehr abbilden. Zur Design-Zeit eines solchen Systems sind alle möglichen Kombinationen von Kontextparametern nicht exhaustiv zu testen, die Gefahr ungültiger oder nicht abzubildender Kombinationen bleibt ein unlösbares Problem solcher Systeme [Tra05].

Ein weiteres Problem bei der kontextuellen Anpassung von Systemen zeigt sich bei der Reversibilität solcher Anpassungen. Je mehr Parameter bei der Anpassung eine Rolle spielen, desto schwerer wird es nachzuvollziehen, welche Kontexteigenschaft nun ursächlich für eine bestimmte Anpassung des Systems war. Als Beispiel diene die Segmentierung eines komplexen Dialoges (z.B. zur Erstellung eines Termins) in Unterdialekt (zur Auswahl der Zeit, dann Eintrag des Betreffs und des Ortes, dann Alarmeinstellungen, usw.). In einem ausschließlich plattformsensitiven System wäre klar, dass die Anpassung aufgrund der Displaygröße erfolgt sein muss, bei einer Änderung der Displaygröße würde sich also auch die Segmentierung des Dialoges verändern, der Zusammenhang ist eindeutig rückverfolgbar. Ist das System nun zusätzlich noch sensitiv gegenüber der kognitiven oder perzeptiven Leistungsfähigkeit des Benutzers, dann kann die Anpassung auch aufgrund der Konzentrationsschwäche des Benutzers erfolgt sein. Dies würde sich nicht zwangsläufig bei einer Änderung der Displaygröße ändern. Zusätzlich ist auch die Problematik der Wechselwirkungen zwischen verschiedenen Parametern ein komplexes Problem, führt die Kombination von Konzentrationsschwäche und kleinem Display zu einer stärkeren Segmentierung, gibt es mehrstufige Anpassungen, oder substituiert die Anpassung an einen Parameter beide?

Aufbauend auf dem oben geschilderten Problem ergibt sich ein weiteres. Ist die Rückverfolgung von Anpassungen nicht möglich, dann kann eine Bearbeitung von kontextadaptiven System auf höheren Abstraktionsebenen nicht erfolgen. In dem Beispiel oben bedeutet dies: kann nicht klar bestimmt werden aus welcher Anpassungsregel die Darstellung resultierte, dann kann eine Veränderung in der Darstellung durch den Entwickler, d. h. eine Bearbeitung in einem graphischen Bearbeitungswerkzeug im Stile eines Visual Designer, nicht realisiert werden. Hier kommen sowohl die Mapping- wie auch die Reversibilitätsproblematik voll zum tragen.

Die Diskussion zeigt, wie umfassend die Problematik der Kontextanpassung von Benutzerschnittstellen ist. Es wurde versucht, Wege aufzuzeigen, in wie weit das Darstellungssystem diese Problematik adressieren kann, und wie eine Ordnung von Entwicklungspfaden zu einer Reduzierung der Komplexität beitragen kann. Eine Vertiefung dieser Thematik liegt jedoch ausserhalb des Rahmens

dieser Arbeit, weswegen an dieser Stelle auf die Arbeit von Dey und Abowd [DA04] verwiesen werden soll, welche eine umfassende Übersicht des Standes der Technik bieten.

6.1.8 Zusammenfassung

Die in diesem Abschnitt vorgestellte Darstellungsplattform dient als prototypische Realisierung des Entwicklungsmodells, welches in Abschnitt 5 erarbeitet wurde. Ziel war es, zu zeigen, dass sich der benutzerzentrierte Entwicklungsprozess bereits in der Architektur des Entwicklungswerkzeugs widerspiegeln kann und sollte. Der Entwickler als Benutzer dieses Entwicklungssystems steht im Zentrum der Überlegungen welche in dem Darstellungssystem umgesetzt wurden.

Das Darstellungssystem dient als Plattform, welche die Umsetzung der Konzepte der sequenziellen Anpassung, der hierarchischen Entwicklungspfade und der transformationalen Vorgehensweise ermöglichen soll. Diese Konzepte liegen in ihren wesentlichen Teilen bereits in der vorliegenden prototypischen Realisierungsform als Eigenschaften des Systems vor und zeigen damit deren Umsetzbarkeit. Der Schwerpunkt der Umsetzung liegt hierbei auf den Konzepten der sequentiellen Anpassung, die auch in dem EMBASSI System (siehe Abschnitt 7.1.1) zur Anwendung kommen, während die hierarchischen Entwicklungspfade nur in Ansätzen realisiert werden konnten. Das Darstellungssystem konnte als Testsystem in verschiedenem Funktionsumfang für drei verschiedene Plattformen des *Microsoft .Net Frameworks* umgesetzt werden.

Als wesentliche Aspekte des Darstellungssystem sind zum einen die Kapselung der komplexen Anpassungsprozesse durch eine gängige Toolkit Programmierschnittstelle und zum anderen die Umsetzung des XFORMS Standards zu nennen.

Die Kapselung führt zu einer Reduzierung des Einarbeitungsaufwandes für den Entwicklung und ermöglicht es im in gewohnter Weise eine Toolkit API in seine Programme einzubinden. Wünschenswert wäre hierbei eine noch weiter gehende Anpassung der API an die Standard WinForms API, so dass diese frei austauschbar sind. Zugleich soll in jedem Fall die volle Kontrolle durch eine zweite, komplexe API zur Verfügung gestellt werden.

Die Verwendung des XFORMS Standards hat neben der Befriedigung der Anforderungen nach Standardkonformität noch den wesentlichen Vorteil, dass die strenge Trennung von Datenmodell und Darstellung, wie sie hier vorgesehen ist, verschiedene Einsatzszenarien ermöglicht, und so neben dem Einsatz auf verschiedenen Endgeräten auch der Einsatz in verschiedenen Topologien möglich ist.

Die erfolgreiche Umsetzung des Entwicklungskonzepts in einem toolkitbasierten Darstellungssystem soll nun als Grundlage für die weitere Betrachtung der graphisch-interaktiven Gestaltungsprozesses dienen. Das Darstellungssystem kann in erster Linie zur Integration plattformübergreifender Oberflächen in Anwendungen genutzt werden und die einfache Übertragung gewährleisten. Wie soll jedoch die Darstellung angepasst werden? Im folgenden Abschnitt werden Methoden zur Unterstützung des Designprozesses diskutiert und damit der Entwicklungsprozess von der entgegengesetzten Perspektive betrachtet werden.

6.2 Design-Unterstützung

Die Möglichkeit, die konkrete Darstellung graphisch-interaktiv in einem *Visual Designer* zu verändern führt zu einer erheblichen Beschleunigung des Entwicklungsprozesses und ermöglicht es auch anderen am Entwicklungsprozess beteiligten Personen, wie Usability Experten und Designern ohne Programmierkenntnisse, in der Gestaltungsprozess aktiv einzugreifen. Die direkte Rückmeldung über das Ergebnis von gestalterischen Änderungen vereinfacht den Prozess und ermöglicht die Fokussierung auf die Darstellung selbst und nicht auf deren technische Umsetzung. Die Gestaltung von Oberflächen mittels solcher graphischer Werkzeuge hat sich aus diesem Grund zu einer der gängigsten Methoden der Oberflächengestaltung entwickelt.

Wie Reiterer [Rei00] feststellt, ist es für die flexible Entwicklung benutzbarer Oberflächen wichtig, „to discover helpful, unobstrusive, structured, and organized ways to integrate the use of principles, guidelines, standards, style guides, and design rules into the design process without stifling creativity.“ Da die Bereitstellung von Informationen wie Style Guides oder allgemeinen Guidelines, auch die Einbindung analytischer Werkzeuge, wie SHERLOCK [MS95] oder auch dem ISO 9241-EVALUATOR [OR97], meist komplementär zum Arbeitsprozess erfolgt, in diesen jedoch nicht integriert ist, müssen Methoden gefunden werden, relevante Maße direkt in Gestaltungshinweise innerhalb des Entwicklungsprozesses umzusetzen.

In diesem Abschnitt werden nun Konzepte beschrieben, wie die Berücksichtigung der *Konsistenz*, und hier insbesondere der *plattformübergreifenden Konsistenz*, in den graphisch-interaktiven Entwicklungsprozess eingebunden werden können. Basierend auf dem Transformationsparadigma (s. Abschnitt 5.2.3) und dem daraus abgeleiteten Konsistenzmaß (s. Abschnitt 5.4) kann die Gestaltung direkt evaluiert werden und das Ergebnis dem Entwickler zurückgemeldet werden. Weiterhin können die zugrundeliegenden Konzepte für die Bereitstellung von Unterstützungsmethoden bei der Gestaltung verwendet werden.

Die konsistente Gestaltung der Benutzerschnittstelle auch innerhalb einer Anwendung ein wichtiger Aspekt bei der Verbesserung der Benutzbarkeit ist (s. Abschnitt 3.3). Sie ist weiterhin eine Voraussetzung für den effektiven Einsatz des transformationalen Konsistenzmaßes. Aus diesem Grund wird in diesem Abschnitt zunächst auf Methoden zur konsistenten Gestaltung von Dialogen generell eingegangen, bevor auf die Unterstützung bei der plattformübergreifenden Entwicklung eingegangen wird.

Die hier beschriebenen Unterstützungsmethoden sind allgemein anwendbar und beziehen sich nicht ausschließlich auf das in Abschnitt 6.1 beschriebene System. Aus diesem Grunde ist es auch durchaus sinnvoll, eine Umsetzung dieser Methoden als Bestandteil gängiger Entwicklungsumgebungen, wie dem *Microsoft Visual Studio*, der *NetBeans IDE* von *Sun Microsystems* oder anderen, in Betracht zu ziehen.

6.2.1 Spezifische Anforderungen

Um eine Entwicklung konsistenter Benutzungsschnittstellen zu ermöglichen, müssen heutige Entwicklungsumgebungen erweitert werden. Eine Entwicklungsumgebung sollte zum einen eine anwendungsübergreifende Perspektive auf die Benutzungsschnittstelle zulassen, um den Abgleich zwischen den Dialogen zu ermöglichen, welcher eine konsistente Darstellung letztlich ausmacht. Zum anderen muss die Möglichkeit geschaffen werden dynamisch zu Laufzeit generierte Elemente zu berücksichtigen um deren Effekt auf die Darstellung beurteilen zu können.

Die *anwendungsübergreifende Perspektive* auf die Ressourcen ist notwendig, um zwischen den verschiedenen Dialogen einer Anwendung vergleichen zu können. Derzeit werden in gängigen Entwicklungsumgebungen alle Ressourcen zwar als Teil eines Projektes, aber inhaltlich unabhängig und getrennt betrachtet. Jeder Dialog, der angelegt wird, wird heute als neuer Dialog von Grund auf neu aufgebaut. Notwendig ist es jedoch einen neuen Dialog als Teil der Dialoge einer Anwendung zu betrachten und die Möglichkeit der Vergleiche in die Umgebung und deren Datenmodell zu integrieren.

Die *Darstellung dynamisch erzeugter Elemente* ist notwendig, weil auch diese Elemente Einfluss auf die Darstellung haben, heute aber nicht in graphisch-interaktiven Gestaltung berücksichtigt werden können. Bei der Entwicklung von Anwendungsoberflächen werden Elemente dann dynamisch erzeugt, wenn deren Anzahl und Ausprägung abhängig von Laufzeitparametern ist. So kann zum Beispiel eine Anwendung abhängig von den Zugriffsrechten des Benutzers Funktionen ein- und ausblenden oder die Anzahl von Schaltflächen abhängig von der Anzahl der zur Laufzeit verfügbaren Module variieren. Die WYSIWYG-Bearbeitung solcher Elemente ist komplex und wird in der Regel nicht unterstützt. Dennoch tragen diese Elemente zum Gesamtbild einer Oberfläche bei und müssen bei der Bewertung zu berücksichtigen sein. Soll die Konsistenzbewertung deshalb möglich sein, muss eine Entwicklungsumgebung Methoden zur Verfügung stellen, wie diese dynamischen Elemente, zum Beispiel durch die Erzeugung von beispielhaften Platzhaltern über einen definierten Wertebereich des ausschlaggebenden Merkmales, bereits im Designprozess darstellbar werden.

Um die plattformübergreifende Konsistenz in den Entwicklungsprozess mit einbeziehen zu können, muss die Entwicklungsumgebung zusätzlich in der Lage sein, zum einen den Programmcode für *verschiedene Plattformen* zu verarbeiten zu können und zum anderen diesen dann in der plattformspezifischen Weise darzustellen. Die Darstellung für verschiedene Plattformen muss hierbei synchron erfolgen und in allen Varianten der Umgebung zur Abfrage und Modifikation von Parametern zugänglich sein. Zu guter Letzt muss die Umgebung die notwendigen Werte zugänglich machen können.

Wie die Übersicht in Abschnitt 4 zeigt, werden diese spezifischen Anforderungen von keiner der existierenden Entwicklungsumgebungen erfüllt. Von den kommerziell erhältlichen Entwicklungsumgebungen ist insbesondere das *Visual Studio .Net* von *Microsoft* am weitesten entwickelt in Bezug auf die geschilderten Anforderungen, so kann hier zwar für verschiedene Endgeräte entwickelt werden (*.Net Framework* für PC, *.Net Compact Framework* für PDA und Smartphones) jedoch kann diese Entwicklung nicht synchron in demselben Projekt erfolgen. Die anwendungsweite Sicht ist ebenfalls nicht möglich, sowie es keine Möglichkeit für die graphische Bearbeitung dynamischer Elemente gibt.

6.2.2 Unterstützung konsistenten Designs

Wie in Abschnitt 5.4 beschrieben wurde, ist konsistentes Design der primären Darstellung eine notwendige Voraussetzung für die Erhebung des Maßes für plattformübergreifende Konsistenz. Soll festgestellt werden, wie konsistent die Abbildung der Position zwischen den Plattformen erfolgte, muss entsprechend auf der primären Plattform eine Regelmäßigkeit erkennbar sein, die Darstellung muss konsistent sein.

Das Werkzeug SHERLOCK von Mahajan und Shneiderman [MS97, Mah96, SCJS95, MS95] ist bislang eine der wenigen Anwendungen, welche die Konsistenz einer Anwendungsoberfläche automatisch evaluieren (s. Abschnitt 4.4). SHERLOCK bietet zwar Methoden zur Evaluierung, nicht aber Konzepte, wie diese Methoden in den Entwicklungsprozess enger eingebunden werden können. Mahajan und Shneiderman konzentrieren sich bei der Unterstützung vielmehr auf die Rückmeldung des Ergebnisses als um die prozessorientierte Unterstützung des Designs selbst. Häufig werden hierbei Zusammenfassungen und Durchschnittswerte berechnet, die bei der individuellen Verbesserung einzelner Elemente nicht direkt eingesetzt werden können. In diesem Abschnitt werden Konzepte beschrieben, wie diese Methoden in einen Entwicklungsprozess eingebunden werden können.

6.2.2.1 Position und Abmessung

In einer graphischen Oberfläche ist die Anordnung der Elemente, also das Layout, eines der wichtigsten Gestaltungsmerkmale. Neben der Optimierung der Anordnung hinsichtlich des Aufgabenflusses (Angemessenheit, [Sea93]) und der inhaltlich sinnvollen Gruppierung der Elemente, wird der Konsistenz der Anordnung eine wichtige Rolle beigemessen (s. Abschnitt 3.3) [Joh00]. Die konsistente Anordnung der Elemente einer Oberfläche betrifft zum einen die *Positionierung* (z.B. der *OK-Button* ist immer unten rechts) und zum anderen die *Größe* (z.B. alle Schaltflächen haben dieselbe Abmessung).

Das SHERLOCK Tool betrachtet Größe und Position in verschiedenen Teilen seines Systems, die einzelnen Aspekte werden hier zusammengefasst und neu kategorisiert.

- *Rasterung (Gridedness)*: die Anzahl der verschiedenen X und Y Positionen der Elemente in einem Dialog. Je stärker das Layout einem Raster folgt, desto geringer ist dieser Wert.
- *Randabstand (Margin Analyzer), Margins*: wertet die Abstände aller äußeren Elemente vom Dialogrand aus und bestimmt die häufigsten Werte als die Angenommenen Optima.
- *Dimensionen (Seitenverhältnis (Aspect Ratio), Button Layout Table)*: Fasst die Aspekte der Konsistenz der Dimension, d. h. der Höhen- und Breitenabmessung zusammen.
- *Relative Positionierung*: analysiert Gruppierungen von Elementen indem die Nachbarschaften nach Häufigkeit gewichtet werden, so können sowohl Gruppen von Schaltflächen (z.B. die typische *OK-Abbrechen-Hilfe* Gruppe) als auch andere relative räumliche Konkordanzan aufgedeckt und eingefordert werden.

Verschiedene Konzepte, dies in den graphisch-interaktiven Gestaltungsprozess einzubinden sind hierbei vorstellbar. In den folgenden Abschnitte werden einige Konzepte vorgestellt.

Konsistenzlinien Eine Möglichkeit die Positionierung von Elementen einheitlich zu gestalten, besteht darin ein Raster oder Hilfslinien vorzugeben an dem sich die Elemente anordnen können. Diese Form der *Snap-Grids* wird in den meisten graphischen Interface Buildern heute bereits angeboten. Nachteil dieser Methode ist es jedoch, dass diese Raster kontextfrei sind, d. h. sie können keinen Bezug zu anderen Elementen in demselben Dialog, geschweige denn in anderen Dialogen herstellen.

Die Möglichkeit der Ausrichtung und Anordnung von Elementen zueinander wird heute meist über gesonderte Methoden realisiert, die in gesonderten Dialogen aufgerufen werden können und die Anordnung entlang einer Achse mit bestimmten Abständen ermöglichen. Da diese Methode den Interaktionsfluss nicht direkt unterstützt sondern diesen vielmehr unterbricht, bieten manche Interface Builder [App05] dynamische Ausrichtungslinien an. Abbildung 6.25 zeigt die verschiedenen Methoden: links oben die dynamischen Ausrichtungslinien, rechts den Dialog zur Anordnung der Elemente und links unten die Hilfslinien.

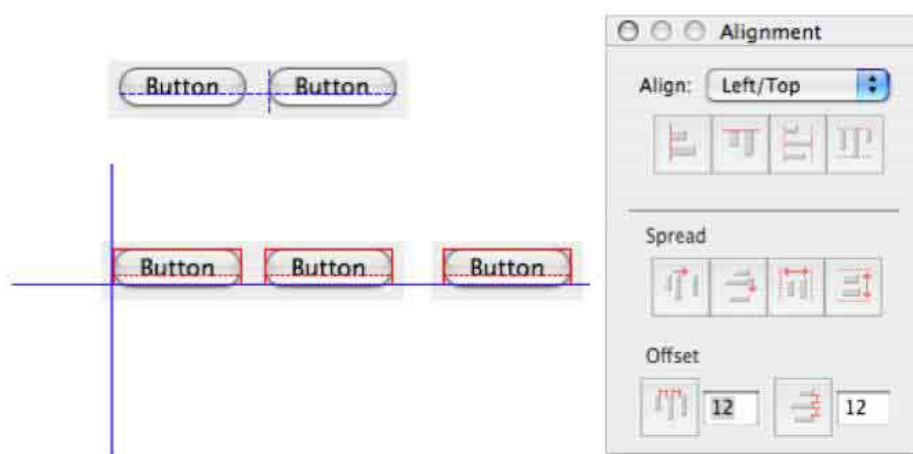


Abbildung 6.25: Beispiel für Design Unterstützung im Apple Interface Builder 2.4.

Wie kann dieser Prozess durch eine anwendungsweite Perspektive unterstützt werden? Abbildung 6.25 zeigt ein Beispiel, wie *Konsistenzlinien* [Ric06] genutzt werden können um anwendungsweit konsistente Positionierung zu unterstützen. Der Entwickler möchte einem Dialog eine neue Schaltfläche („Open“) hinzufügen. Hierzu zieht er aus einem Elemente-Baukasten eine Schaltfläche in den Dialog und positioniert sie dort über *Drag & Drop*. Über die dynamischen Hilfslinien kann er wie gehabt das Element an anderen Elementen in demselben Dialog ausrichten. Sind die Konsistenzlinien aktiv, werden in der unmittelbaren Umgebung der aktuellen Position des Elements alle ähnlichen

Elemente in anderen Dialogen der Anwendung angezeigt und können so zur Ausrichtung genutzt werden.

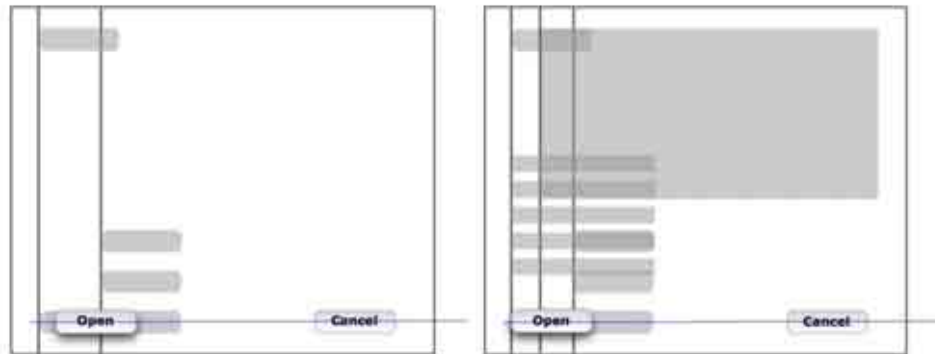


Abbildung 6.26: Beispiel für vertikale Konsistenzlinien.

Abbildung 6.26 zeigt, wie dies in der Praxis umgesetzt werden kann. Die vertikalen Linien stehen für den linken Rand anderer Elemente auf anderen Dialogen der Anwendung. Der Übersichtlichkeit wegen sollten nur die Linien in unmittelbarer Nähe des aktuellen Elements angezeigt werden. Weiterhin werden die Silhouetten der Referenzobjekte angezeigt um so einen Eindruck von der räumlichen Verteilung zu erhalten. Es ist hierbei sinnvoll sich hier auf dezente Silhouetten anstatt der kompletten Objekte zu beschränken um die Ansicht nicht zu überladen.

Als Referenz können entweder nur Elemente desselben Typs (linke Seite) oder alle verfügbaren Elemente (rechte Seite) angezeigt werden. Die Silhouetten haben auf die Positionierung denselben Einfluss wie ein *Snap-Grid*, d. h. ein Element wird davon angezogen, wenn es in unmittelbarer Nähe losgelassen wird. Auf diese Weise kann eine konsistente Positionierung aller Elemente über die verschiedenen Dialoge einer Anwendung hinweg erreicht werden. Die Ausrichtung kann hierbei ebenfalls in horizontaler Richtung oder in beide Richtungen zugleich möglich sein. Ebenso kann zwischen dem rechten und linken Rand der Referenzobjekte variiert werden.

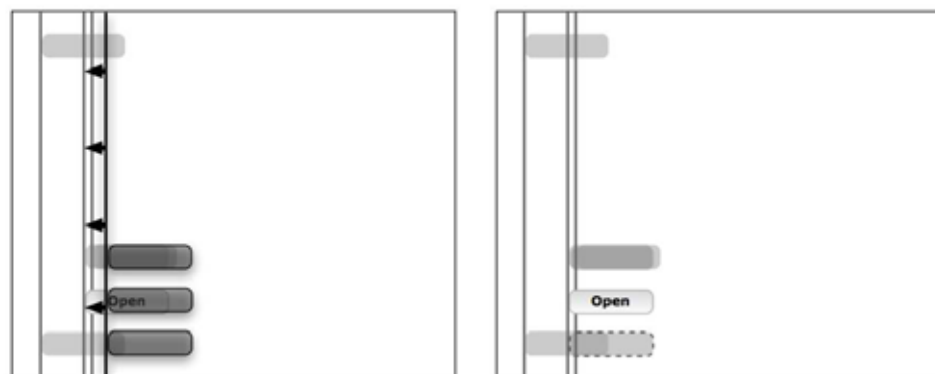


Abbildung 6.27: Die Konsistenzlinien können genutzt werden um Elemente zusammenzuführen.

Die Anzahl und Anordnung der Konsistenzlinien kann auch zur Beurteilung der Rasterung herangezogen werden, treten sie für ähnliche Elemente in engen Abständen auf, spricht dies für eine geringe Rasterung. Abbildung 6.27 zeigt, wie die Konsistenzlinien umgekehrt auch genutzt werden können, um die Rasterung zu verbessern. Elemente mit ähnlichen Positionen können hier z. B. auch über ein Verschieben der Konsistenzlinie über alle Dialoge hinweg als Gruppe verschoben werden (linke Seite) und so die Rasterung verbessert werden (rechte Seite).

Konflikte durch Überlappung müssen hierbei berücksichtigt werden. In Abbildung 6.27, rechte Seite wird eine Silhouette mit gestrichelter Kontur gezeigt (Farben wären eine andere Möglichkeit),

dies deutet auf einen Konflikt hin. Durch direktes Anklicken der Silhouette sollte der Wechsel zu dem betreffenden Dialog ermöglicht werden. Weiterhin sollten Elemente explizit aus der Rasterung ausgeschlossen oder generell fixiert werden können um diese bei der Verschiebung auszusparen.

Ähnlich wie bei der Positionierung können die Konsistenzlinien auch zur Vereinheitlichung der Abmessungen der Elemente genutzt werden.

Es kann vermutet werden, dass das Verfahren der Konsistenzlinien insbesondere bei Dialogen mit vielen kleinen Elementen, so wie formularbasierte Eingabemasken, den interaktiven Designprozess verbessern kann.

Dynamische Vorlagen Die Verwendung von Vorlagen bei der Gestaltung von digitalen Medien, zu denen Dialoge ebenfalls gezählt werden können, hat bereit eine lange Tradition. Dokumentverarbeitungssysteme wie die Office Anwendungen (*Microsoft Word*, *Microsoft Powerpoint*, *Staroffice*, etc.) bieten alle die Möglichkeit an, Vorlagen zu definieren. Basierend auf diesen Vorlagen können neue Dokumente erstellt werden. Vorteil hiervon ist die einheitliche Gestaltung und die Möglichkeit Layout-Änderungen auf einen ganzen Satz von Dokumenten allein durch die Änderung der Vorlage zu erledigen.

Nichols *et al.* [NML04] haben die Verwendung von Vorlagen auf die Entwicklung von plattformunabhängigen Benutzerschnittstellen übertragen (s. Abschnitt 4.3.3.1 ff.). Hier werden abstrakte *Smart Templates* definiert, welche die Anordnung und Auslassung von Elementen je nach Plattform beschreiben und dabei dennoch eine konsistente Darstellung ermöglichen sollen.

Die Verwendung von Vorlagen bei der graphischen Gestaltung von Benutzerschnittstellen in Entwicklungsumgebungen ist bislang in der Regel anwendungsspezifischen Werkzeugen zur Erzeugung von Datenbankoberflächen oder Websites, etc. vorbehalten. In diesen Fällen werden meist feste Vorlagen definiert und dann vom Designer entsprechend der Anforderungen gewählt.

Dynamische Vorlagen [Ric06] könnten der Generalisierbarkeit dieses Ansatz Vorschub leisten, indem sie sich aus der Anwendung heraus generieren und gemeinsam mit der Anwendung anpassen lassen können. Abbildung 6.28 zeigt, wie aus einem Dialog für eine Anwendung eine Vorlage aus einem Vorlagen Prototyp angelegt werden kann. Diese Vorlage kann dann bei der Erzeugung neuer Dialoge als Vorlage verwendet werden. Eine einheitliche Gestaltung ist somit sichergestellt.

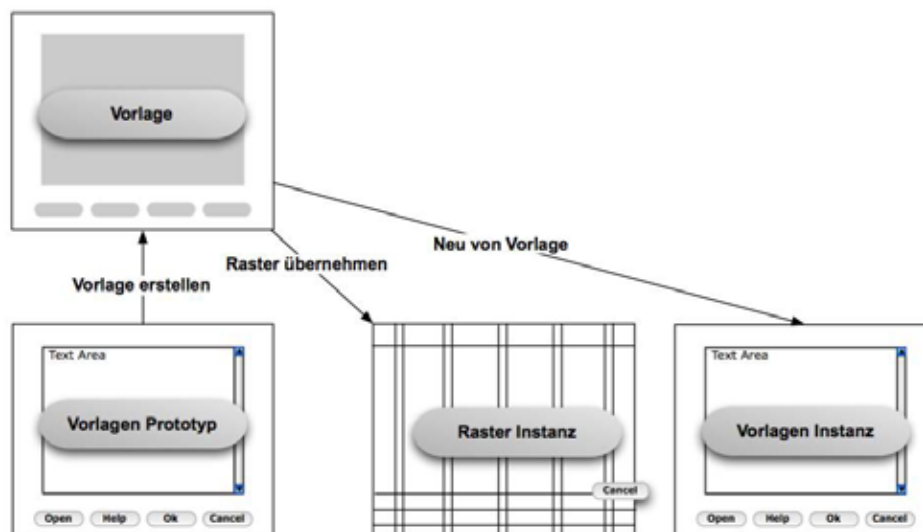


Abbildung 6.28: Ablauf der Erstellung und Verwendung dynamischer Templates

Um Vorlagen wirklich dynamisch zu gestalten, sollte der Vorlagenkatalog aus den vorhandenen Dialogen automatisch extrahiert werden. Alle Dialoge einer Anwendung stehen als Prototypen für Vorlagen zur Verfügung. Der Designer kann also beim Anlegen eines neuen Dialogs bestimmen, dass

dieser *aussehen soll wie Dialog x*. Bei Veränderungen im Vorlagenprototyp muss dann vom Designer entschieden werden, ob diese auf die Vorlage übertragen werden soll und damit alle anderen Instanzen dieser Vorlage betreffen soll, oder ob die Vorlage von dem Dialog abgekoppelt werden soll. Ähnliche Dialoge, oder Dialoge, die von einer Vorlage abgeleitet wurden, werden im Vorlagenkatalog nicht als unabhängige Prototypen sondern als Instanzen angezeigt.

Dieses Konzept dynamischer Vorlagen stellt eine komplexere Ableitung derselben Prinzipien wie bei den Konsistenzlinien dar. Die anwendungsweite Perspektive ermöglicht die Vereinheitlichung der Position und Dimension einzelner Elemente über die gesamte Anwendung. Die Verbesserung der Konsistenz der Darstellung kann genutzt werden um die graphisch-interaktive Entwicklung benutzbarer Anwendungsoberflächen zu unterstützen.

6.2.2.2 Farbgebung und Typographie

Bei der visuellen Wahrnehmung spielt die Farbgebung eine wichtige Rolle bei der Orientierung innerhalb eines Dialoges. Grobstrukturen, Gruppierungen aber auch die Aufmerksamkeitslenkung durch das Setzen farblicher Akzente sind Mittel des gezielten Einsatzes von Farben [Sch05]. Andererseits besteht aber auch leicht die Gefahr ein Design durch unpassende und inkonsistente Farbgebung zu verschlechtern [GL05].

In dem SHERLOCK Tool wird die Farbgebung in zwei Funktionen analysiert. Abweichungen der Hintergrund- bzw. Vordergrundfarben werden identifiziert und als Teil des *Button Concordance Tool* speziell Farbgebung die der Schaltflächen untersucht. Eine interaktive Einbeziehung in den Designprozess ist auch hier nicht umgesetzt.

Ähnlich wie bei der Farbgebung hält die typographische Gestaltung der textuellen Inhalte die Möglichkeit der Akzentuierung und Strukturierung bereit. Ebenso wie dort führt auch die inkonsistente Verwendung zu Verwirrung und zu einer Abwertung des Designs [MS97, Joh00, GL05].

SHERLOCK analysiert *Abweichungen des Schriftsatzes (Distinct Typefaces)* und gibt diese in der Statistik der Übersichtstabelle an.

Insbesondere bei den gestalterischen Dimensionen der Typographie und der Farbe besteht das Problem meist darin, dass hier über die Gestaltung inhaltliche Bedeutung transportiert und unterstützt werden soll. Ist ein Hinweis beispielsweise wichtig, wird er in fetter Schrift dargestellt oder ist ein Fehler kritisch, wird er rot dargestellt.



Abbildung 6.29: Dialog zur Bestimmung einer Formatvorlage in Microsoft Word.

Obgleich diese Vorgehensweise bei dokumentorientierten Anwendungen (z. B. Microsoft Word, s. Abbildung 6.29) in der Zwischenzeit gängige Praxis ist, hat sich dieser Ansatz nicht im Design interaktiver Systeme durchsetzen können. In den heutigen Entwicklungsumgebungen gibt es weder die Möglichkeit, Elementen eine Bedeutung zuzuordnen (*wichtig, kritisch*) noch farbliche oder typographische Informationen mit Elementen solcher Bedeutungen zu verknüpfen (z. B. *stelle alle kritischen Fehler rot dar*).

Allerdings verfügen einige UIMS und Beschreibungssprachen wie in Abschnitt 4 beschrieben, die Möglichkeit Darstellungsattribute über abstrakte Begriffe zuzuweisen. Auch in der hier beschriebenen Darstellungsplattform in Abschnitt 6.1 wird die Darstellung entsprechend des Prinzips der Klassen aus den *Cascading Style Sheets (CSS)* in dieser Weise umgesetzt. Der Vorteil dieses Ansatzes ist es, dass sich die Bedeutung eines Elementes nicht ändert, die Darstellung hingegen u. U. angepasst werden muss. Durch Eingriff in die Vorlage werden alle Instanzen zugleich angepasst und die Darstellung bleibt konsistent.

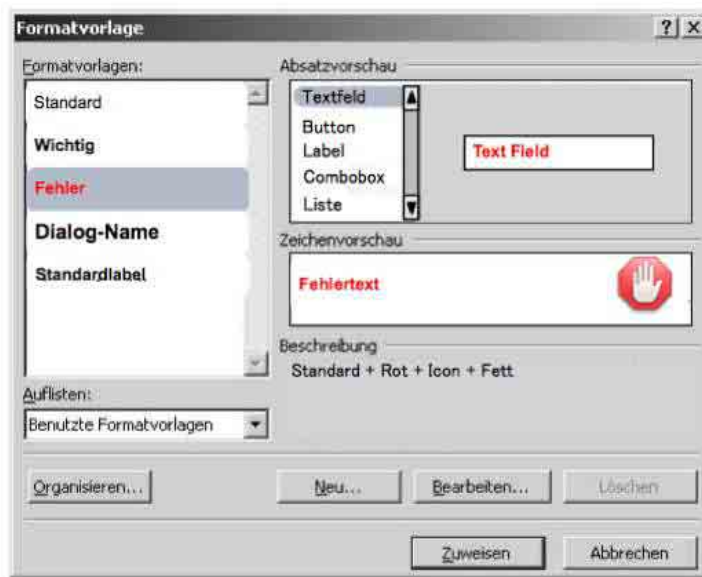


Abbildung 6.30: Beispiel für Formatvorlage bei der UI-Entwicklung.

Abbildung 6.30 zeigt, wie die Auswahl der Formatvorlage in einem Dialog analog zur Dokumentbearbeitung erfolgen kann. Verschiedene Attribute können hierbei spezifisch für bestimmte Elemente definiert werden. Andere Attribute wie Farbgebung und Textsatz können generell für alle Elemente einer Bedeutungsgruppe definiert werden. In der linken Liste in Abbildung 6.30 werden alle Vorlagen angezeigt, wählt man diese aus, erhält man eine Beschreibung und Vorschau. Rechts oben kann dann eine elementspezifische Vorschau gewählt werden.

6.2.2.3 Terminologie

Die textuelle Kohärenz der Beschriftungen spielt eine wichtige Rolle in der Bedienbarkeit. So soll nicht nur die sprachliche Ebene des Benutzers eingenommen werden, sondern Informationen in einheitlicher Weise dargeboten werden um Missverständnisse zu vermeiden und die Lernbarkeit zu verbessern. Auch der Austausch über eine Anwendung kann so unterstützt werden, so dass es einfacher ist die Bedienung zu erklären. Dieser Aspekt ist insbesondere bei der Schulung, aber auch bei der Gestaltung von Bedienungsanleitungen wichtig. Einige bemerkenswerte Beispiele textueller Fehlentwürfe findet man bei Johnson [Joh00].

In SHERLOCK werden hierzu die sogenannten *Terminology Bags* gebildet, die semantisch ähnliche Begriffe enthalten und synonym verwendet werden können. Das Auftreten von Synonymen wird angezeigt und die Verwendung eines einheitlichen Begriffs vorgeschlagen. Das *Interface Concordance Tool* sammelt ebenfalls alle Beschriftungen aus den Dialogen und zeigt Unterschiede in der Nutzung von Groß- und Kleinschreibung sowie Abkürzungen auf. Das *Concordance Tool* sammelt die textuellen Inhalte, insbesondere in der Kurzbeschreibungen und sortiert diese nach Ähnlichkeit. Der Sinn hiervon ist es, leichte Variationen in der Benennung aufzuzeigen und den Entwickler bei der konsistenten Benutzung von Beschriftungen zu unterstützen.

Terminology Bags Die *Terminology Bags* des SHERLOCK Systems enthalten Synonyme für bestimmte Funktionsbezeichnungen. Werden für Elementbeschriftungen verschiedene Bezeichnungen aus derselben Synonymgruppe verwendet, wird das als inkonsistente Bezeichnung gewertet. Problem hierbei ist, dass die Synonymgruppen statisch definiert werden müssen. Bestimmte Worte können nicht in dem Korpus bestimmt sein und Inkonsistenzen nicht aufgedeckt werden.

Die Modellierung auf Basis von Aufgabenmodellen stellt einen Ansatz dar, wie dieses Problem durch Spezifizierung der Funktion eines Elements gelöst werden kann [PLV01].

Eine weitere Lösung dieses Problems besteht darin, dass für eine Anwendung textuelle Inhalte an einer Stelle definiert werden können und nur aus diesen dann bei der Erzeugung neuer Elemente die textuelle Beschreibung ausgewählt werden. Ein anwendungsweites Textressourcenmanagement in benutzerdefinierten *Terminology Bags* kann als möglicher Lösungsansatz genutzt werden. Verschiedene Beschreibungssprachen wie z. B. UIML [AP99] oder die ANSI INCITS 389ff. Familie [Int05a] (s. Abschnitt 4.3.3.1) verfügen über eine solche Absonderung von textuellen Inhalten. Auch in kommerzieller Software Entwicklung findet die Ausgliederung von sprachabhängigen Inhalten im Rahmen der Internationalisierung von Software statt. Hier werden textuelle Inhalte für verschiedene Sprachen vorgehalten und dann zur Laufzeit, abhängig von der gewählten *Locale* geladen [Küs03]. Es liegt also nahe, diesen Mechanismus im Rahmen der Verbesserung der Konsistenz zu erweitern.

Abbildung 6.31 zeigt beispielhaft wie die Umsetzung des anwendungsweiten Textressourcenmanagements umgesetzt werden könnte. Alle Texte werden in dem Ressourcenmanager verwaltet. Ein Text kann verschiedene lokalisierte Versionen besitzen, welche alle mit derselben Ressource verknüpft sind. Zum Anzeigen wird hier der Text mit der *Locale* der gegenwärtigen Umgebung verwendet. In der linken Spalte kann man wählen, welche Ressourcen man sehen möchte und wie oft diese in welchen Kontexten verwendet wurden, so wurden drei verschiedene Ressourcen in Schaltflächen verwendet. In der mittleren Spalte sieht man die Textressourcen. In der rechten Spalte oben werden die Bezeichnungen in den verschiedenen Sprachen angezeigt und editiert. In der unteren Hälfte wird angezeigt, welche Elemente auf diese Textressource verweisen, hier kann auch direkt auf dieses Element navigiert werden. Das ange deutete Kontextmenü zeigt an, welche Aktionen auf einer Textressource durchgeführt werden.

Neben dem Effekt der Lokalisierung und der Verbesserung der Konsistenz ist dieses Konzept geeignet, Elementen auch reduzierte Texte zuzuweisen. Auf diese Weise können Beschriftungen dann zugleich für mobile Geräte zur Verfügung gestellt werden.

Wortergänzung Wortergänzung (engl. *word completion*) ist ein Verfahren, welches bereits seit langem in vielen Kontexten eingesetzt wird. So bieten z. B. Tabellenkalkulationen wie *Microsoft Excel* Wortergänzung bei der Bearbeitung von Werten in einer Tabelle an, um so Schreibfehler oder

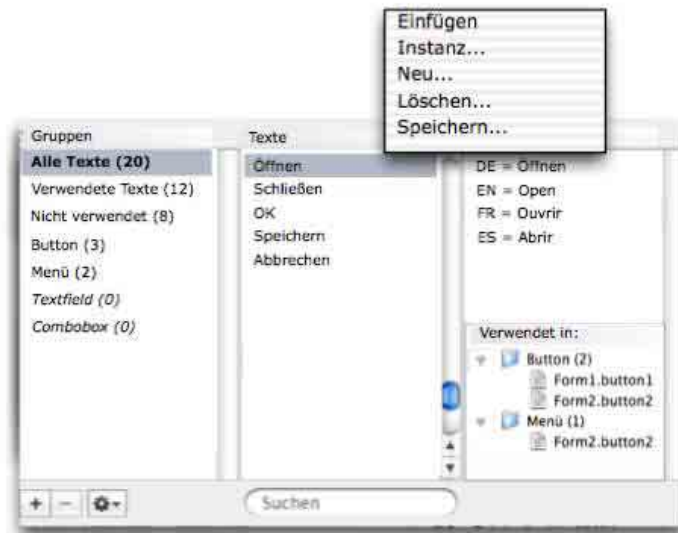


Abbildung 6.31: Beispiel für die Verwaltung von Textressourcen

Variationen in Listen zu verhindern und die kognitive Last des Benutzers zu reduzieren. Dasselbe Verfahren könnte in der Entwicklung von Benutzerschnittstellen eingesetzt werden um die textuelle Konsistenz der Inhalte zu verbessern.



Abbildung 6.32: Beispiel für Wortergänzung

Abbildung 6.32 verdeutlicht die Umsetzung dieses Konzeptes. Bei der Eingabe der Beschriftung eines Buttons werden die Buchstaben 'Öf' eingegeben. Das System sucht in allen Elementen der Anwendung (oder im Textressourcenmanager, s. o.) nach verfügbaren Texten die dem Eingabemuster entsprechen könnten und zeigt diese in einem Popup-Dialog unterhalb der Cursor-Position an. Rechts davon werden im Falle des Vorhandenseins eines Textressourcenmanagers die lokalisierten Begriffe angezeigt (evtl. mit einer zeitlichen Verzögerung).

Die Vorhersage kann zusätzlich verbessert werden, indem Texte, die in dem selben Kontext (Element, Position, Nachbarschaft, o. ä.) verwendet wurden, weiter oben angesiedelt werden. Eine Korrekturhilfe (s. u.) kann zusätzlich ähnliche Worte mit kleinen Abweichungen anzeigen. Hier wird z.B. ebenfalls *Offen* angezeigt. Groß- und Kleinschreibung kann auf diese Weise ebenfalls einfach vereinheitlicht werden.

Korrekturhilfe Soll der Benutzer dabei unterstützt werden ein bestimmtes Vokabular zu verwenden, wird insbesondere in den dokumentorientierten Anwendungen die Korrekturhilfe verwendet. Eine vertraute Metapher kommt hier zum Einsatz, die an das Korrekturverfahren in der Schule oder bei der Rezension erinnert. Begriffe die manuell eingetragen wurden, werden mit einem Wortschatz verglichen und im Falle, dass sie nicht Bestandteil dieses Wortschatzes sind, als fehlerhaft markiert.



Abbildung 6.33: Beispiel für eine Korrekturhilfe in Microsoft Word

Abbildung 6.33 zeigt ein Beispiel aus Microsoft Word. Eine fehlerhafte Eingabe wird rot unterstrichen, durch die gezackte Linie lässt sich ein Fehler deutlich von regulären Unterstreichungen unterscheiden. Über das Kontextmenü kann nun die Korrekturhilfe aufgerufen werden und aus alternativen Vorschläge ausgewählt werden. Über Die Option *Hinzufügen* kann ein Wort explizit als korrekt identifiziert werden und somit in den Wortschatz aufgenommen werden.

Im Falle des graphischen Editors stellt das Lexikon die textuellen Inhalte der Dialoge der Anwendung dar. Um einen neuen Inhalt einzubinden, muss man diesen explizit bestätigen. Auch diese Anwendung lässt sich mit den *Terminology Bags* oder dem Textressourcenmanagement kombinieren.



Abbildung 6.34: Beispiel für eine kontextsensitive Konsistenzprüfung

Kontextspezifische Unterstützung (analog zu einer Grammatiküberprüfung) sind hier ebenfalls denkbar. So kann der Ausdruck *Öffnen* zwar in einer Anwendung bereits einem Button vorgekommen sein, nicht jedoch in einer Liste. Oder die Position eines Elements wird als Kontext hinzugezogen, um die Benennung und die Positionierung konsistent zu halten. Das System zeigt einen möglichen Konflikt an, wenn ein Button mit *OK* beschriftet wird, der an einer Position liegt, an der in anderen Dialogen der Anwendung der *Abbrechen* Button liegt. Abbildung 6.34 zeigt ein Beispiel für eine kontextsensitive Konsistenzprüfung des textuellen Inhaltes.

Alle in diesem Abschnitt beschriebenen Konzepte dienen dazu, die Konsistenz innerhalb einer Anwendung zu verbessern. Ziel ist es hierbei Interaktionskonzepte zu beschreiben, die bei der Gestaltung von Entwicklungsumgebungen eingesetzt werden können, um den Entwickler graphisch-interaktiv und intuitiv bei der konsistenten Gestaltung unterstützen. Dies ist insbesondere dann notwendig, wenn eine Anwendung mehrere Duzend Dialoge umfasst und diese strukturell ähnlich aufgebaut sind. Neben der Möglichkeit abstrakte Oberklassen für bestimmte Dialogfamilien zu definieren, ist dieser graphisch-interaktive Ansatz besonders geeignet, sich übergangslos in die direkt-manipulative Interaktionsmetapher moderner Entwicklungsumgebungen einzupassen.

Der Fokus dieser Arbeit liegt nur sekundär auf den Aspekten der anwendungsinternen Konsistenz. Jedoch wie bei der Konzeption des plattformübergreifenden Konsistenzmaßes festgestellt wurde, ist diese Form der Konsistenz eine notwendige Voraussetzung für die sinnvolle Ableitung konsistenter plattformspezifischer Varianten.

6.2.3 Unterstützung plattformübergreifender Konsistenz

Das plattformübergreifende Konsistenzmaß welches in Abschnitt 5.4 beschrieben wird eignet sich nicht nur zur Evaluierung der Konsistenz, um dann wie bei SHERLOCK (s. Abschnitt 4.4) in summarischen Beschreibungen und Tabellen präsentiert zu werden. Das Konsistenzmaß kann ebenfalls zur Unterstützung des graphisch-interaktiven Entwicklungsprozesses genutzt werden, indem zum einen das Maß als numerischer Wert während des Gestaltungsprozesses selbst erfasst und zur Optimierung genutzt wird. Zum anderen können die kognitiven Prozesse, welche als ursächlich für die Validität des Konsistenzmaßes angesehen werden, Hinweise für andere Konzepte geben, zur Gestaltung von Unterstützungsmethoden genutzt werden.

6.2.3.1 Position und Orientierung

Wie das Beispiel in Abschnitt 5.4.5.2 zeigt, kann die plattformübergreifende Konsistenz auch anhand der Maße der Position und der Orientierung berechnet werden. Ist eine Abbildung diesbezüglich *transformationally consistent*, dann kann aufgrund der Position im Original mit einer Wahrscheinlichkeit von $p = 1.0$ oder 100% die Position des Elements in der Abbildung bestimmt werden. Oder einfacher ausgedrückt: steht ein Element an der Position $P = \{x, y\}$ dann wird es in der Abbildung in n von n Fällen auf Position $P = \{ax, by\}$ stehen.

Eine Beispielanwendung Wie kann dieses Maß nun eingesetzt werden um den graphisch-interaktiven Gestaltungsprozess zu unterstützen? Abbildung 6.35 zeigt eine Beispielanwendung, die bei der Gestaltung konsistenter plattformübergreifender Anwendung Unterstützung geben soll.

Wie in den spezifischen Anforderungen in Abschnitt 6.2.1 beschrieben wurde, sollte eine solche Entwicklungsumgebung eine *anwendungsweite Perspektive* schaffen. Das hier gezeigte System realisiert dies indem alle Dialoge einer Anwendung in derselben Ansicht angezeigt werden. Bei größeren Anzahl von Dialogen sollte dies evtl. durch eine hierarchische Struktur, oder die Möglichkeit zu *zoomen* ergänzt werden.

Das Hauptfeld der Anwendung zeigt die Dialoge an. Auf der linken Seite befinden sich die Dialoge der *primären* Plattform (s. Abschnitt 5.2.2), auf der rechten alle abgeleiteten *sekundären* Plattformen. In diesem Fall ist nur eine sekundäre Plattform eine Pocket PC Anwendung sichtbar. Zwischen den Dialogen liegt jeweils ein *Überdeckungsbild*, bei dem die Umrisse der Elemente der primären und der sekundären Plattform in normalisiertem Maßstab aufeinander projiziert wurden. Auf diese Weise kann die relative Transformation der Position und Dimension dargestellt werden. Zusätzlich zeigt ein Translationsvektor die Verschiebung der Mittelpunkte zwischen primärer und sekundärer Darstellung. Durch Auswahl eines Dialoges kann direkt in die Ansicht des Dialogs gewechselt werden.

Auf der linken Seite ist eine Liste der Konsistenzmaße, die für die plattforminterne Konsistenz der Primärplattform, angezeigt werden sollen. Hier können die Maße aus der SHERLOCK-Toolbox eingefügt werden und durch geeignete Visualisierung dargestellt werden.

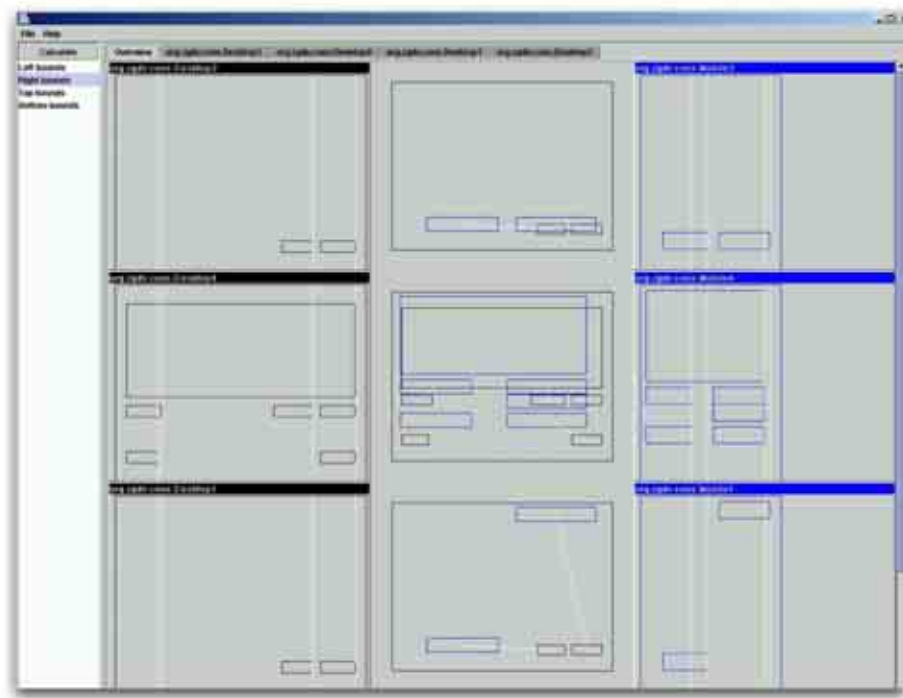


Abbildung 6.35: Beispiel eines Entwicklungswerkzeuges

In dem Beispiel kann die Visualisierung der vertikalen Konsistenzlinien (s. Abschnitt 6.2.2.1) gesehen werden. Weitere Umsetzungen der im vorigen Abschnitt beschriebenen Methoden können an dieser Stelle eingefügt werden.

Positions- und Orientierungskonsistenz Die Evaluierung der Konsistenz benötigt die gleichzeitige Berücksichtigung aller Elemente auf der primären Plattform und deren Gegenüberstellung gegen die entsprechenden Elemente auf der sekundären Plattform. Derzeit beschränkt sich dies auf einen paarweisen Vergleich. Abbildung 6.36 zeigt den Ausschnitt einer Anwendung, welche die Konsistenz der Position und Orientierung zweier Darstellungen evaluiert und das Ergebnis direkt an den Entwickler zurückmeldet.

Die Darstellung erfolgt ebenfalls im Sinne eines Überdeckungsbildes. Die Elemente aller Dialoge der primären und der sekundären Plattform werden, farblich unterschiedlich, als Umrisse eingezeichnet. Die Abmessungen werden hierbei auf die Dimensionen des umgebenden Dialogs normalisiert dargestellt.

$$D = \{W, H\} \quad (6.1)$$

$$E = \{x, y, w, h\} \quad (6.2)$$

$$E' = D/E \quad (6.3)$$

$$= \{(x/W), (y/H), (w/W), (h/H)\} \quad (6.4)$$

Wobei D der umgebende Dialog, E das Element, und $\{W, H\}$ bzw. $\{x, y, w, h\}$ die Position und Abmessung des Dialogs, bzw. Elements darstellen. Die resultierende Darstellung ist zwar verzerrt, erlaubt aber die Visualisierung aller Elemente in demselben Bezugssystem. Auch können so die Translationsvektoren sinnvoll eingesetzt werden.

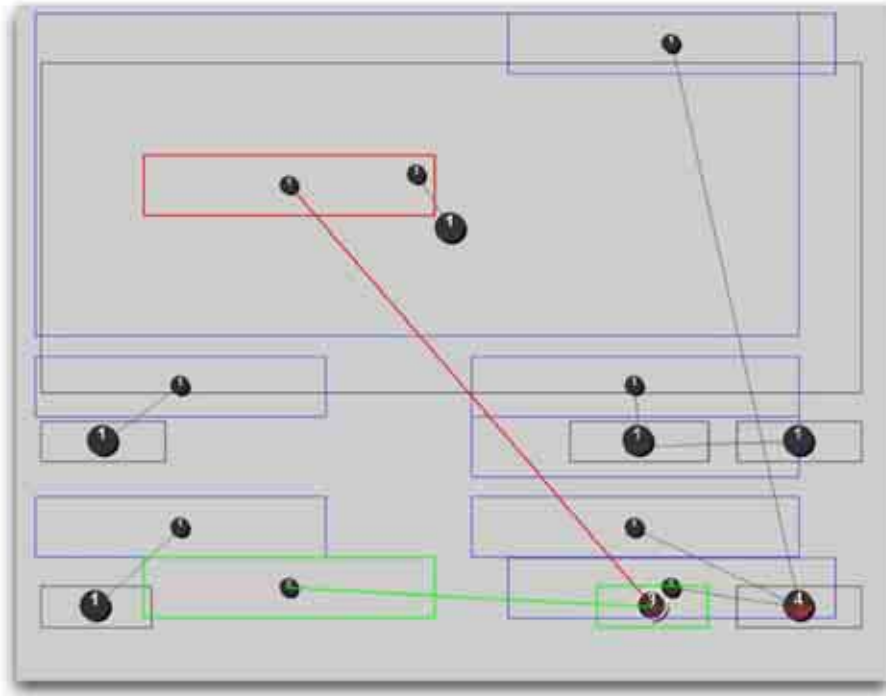


Abbildung 6.36: Beispiel für eine Anwendung zur Unterstützung plattformübergreifenden Designs.

Die Darstellung aller Elemente zugleich ist bei wenigen Elementen bzw. Dialogen noch überschaubar, verliert bei mehr als sechs Dialogen jedoch schnell an Aussagekraft. In diesem Falle sollte bei der Darstellung nach Kriterien wie Element-Typ oder Beschriftung o. ä. zu filtern sein.

Zur Konsistenzberechnung werden nun Positionsgruppen gebildet. Alle Elemente (E) an einer bestimmten Position in der primären Darstellung werden, zuzüglich eines zu definierenden Unschärfefaktors (α, β) in Positionsgruppen gefasst.

$$E \in G \Rightarrow \forall \{x_E, y_E\} = \{\alpha x_G, \beta y_G\} \quad (6.5)$$

Prinzipiell kann für beide Bestandteile $\{x, y\}$ der Position ein Konsistenzmaß berechnet werden. Um die Komplexität zu reduzieren, bietet es sich bei der Entwicklungsunterstützung jedoch an, ein kombiniertes Maß zu verwenden. Dies bedeutet, dass ein Element nur in eine Positionsgruppe fällt, wenn beide Werte innerhalb der Toleranz liegen.

Da Elemente innerhalb eines Dialoges einen eindeutigen Bezeichner besitzen, kann nun für jedes Element der primären Plattform das korrespondierende Element der sekundären Plattform gesucht, und dessen normalisierte Position abgefragt werden. Die Elemente der sekundären Plattform werden dann analog zu Gleichung 6.5 ebenfalls in Positionsgruppen (G') eingeteilt.

Die stochastische Konsistenz lässt sich nun nach Gleichung (5.19) anhand der Größenverhältnisse der sekundären Positionsgruppen berechnen. Die Anzahl der Elemente in der am häufigsten belegten Positionsgruppe muss gegen die Gesamtanzahl der Elemente in Relation gesetzt werden. Nimmt man also G'_{max} als die häufigste Gruppe an, dann ist diese mit $n(G'_{max})$ Elementen besetzt. Die Konsistenz berechnet sich damit wie folgt.

$$C = n(G'_{max})/n \quad (6.6)$$

Sind zwei oder mehrere Gruppen gleich häufig besetzt, dann kann die Konsistenz durch die Übernahme der Anzahl von Elementen in einer dieser Gruppe berechnet werden. Die Wahrscheinlichkeit ist in diesem Fall dieselbe, ganz gleich für welche der häufigsten Gruppen der Benutzer sich entscheidet.

Abbildung 6.36 zeigt nun, wie die Konsistenz anhand der Translationsvektoren und Elementankern visualisiert werden kann. In diesem Beispiel wurde auf den Elementanker der primären Gruppe (dargestellt durch einen Kreis in der Mitte der Elemente dieser Gruppe, kleinere Kreise stehen für die Anker sekundärer Gruppen) geklickt.

Die Konsistenz wird durch die Einfärbungen der Translationsvektoren und der Gruppensilhouetten visualisiert. Grüne Kanten zeigen die konsistente Darstellung an (d. h. die am häufigsten besetzte Gruppe). Rote Kanten zeigen die inkonsistenten Darstellungen. Die Häufigkeitsverhältnisse werden über die Zahlenwerte in den Kreisen ($3 \rightarrow 2$ und $3 \rightarrow 1$; d. h. in der häufigeren Gruppe sind 2 der 3 Elemente abgebildet) dargestellt. Im Falle der gleichen Besetzung von Gruppen wird alles als inkonsistent dargestellt, da sich keine dominante Darstellung herausgebildet hat und damit der Benutzer zum Raten gezwungen ist.

Durch Auswahl der Anker der primären Elementgruppen wird die Konsistenz dargestellt, durch Auswahl der sekundären Anker, wechselt der Benutzer direkt in den Bearbeitungsmodus des sekundären Dialoges. Dies ist insbesondere dann interessant, wenn direkt zu den inkonsistenten Elementen navigiert werden kann. Das Bedienungskonzept unterstützt so den Arbeitsvorgang unmittelbar. Für den Fall dass eine sekundäre Gruppe mit mehreren Elementen besetzt ist, besteht die Möglichkeit, stets zum ersten Element dieser Gruppe zu springen. Alternativ wäre auch der Wechsel in einen Übersichtsmodus, ähnlich Abbildung 6.35 möglich.

Abbildung 6.37 zeigt den Bearbeitungsmodus. In diesem kann der Benutzer Elemente durch *Drag & Drop* neu positionieren.



Abbildung 6.37: Die konsistente Position wird als Silhouette angezeigt.

Die Positionierung wird, analog zu den Bedienkonzepten in Abschnitt 6.2.2.1, durch Anzeigen der Silhouette der häufigsten Elementgruppen und einer *Snap Position* geleitet. In Abbildung 6.37 wird dies durch eine dunkelgraue Silhouette visualisiert. Bei Annäherung an diese Silhouette *rastet* das bewegte Element ein (*Snap Position*). Innerhalb eines bestimmten Umkreises wird die Position also automatisch gleich der Position der konsistenten Darstellung gesetzt. Dadurch wird der feinmotorische Aufwand bei der Positionierung reduziert, ein Konzept, welches in vielen gängigen graphischen Werkzeugen Anwendung findet.

Den Prinzipien der direkt-manipulativen Interaktion folgend [Shn98], gibt das System unmittelbare Rückmeldung über das Resultat der Handlung. In Abbildung 6.38 wird gezeigt, wie die Visualisie-

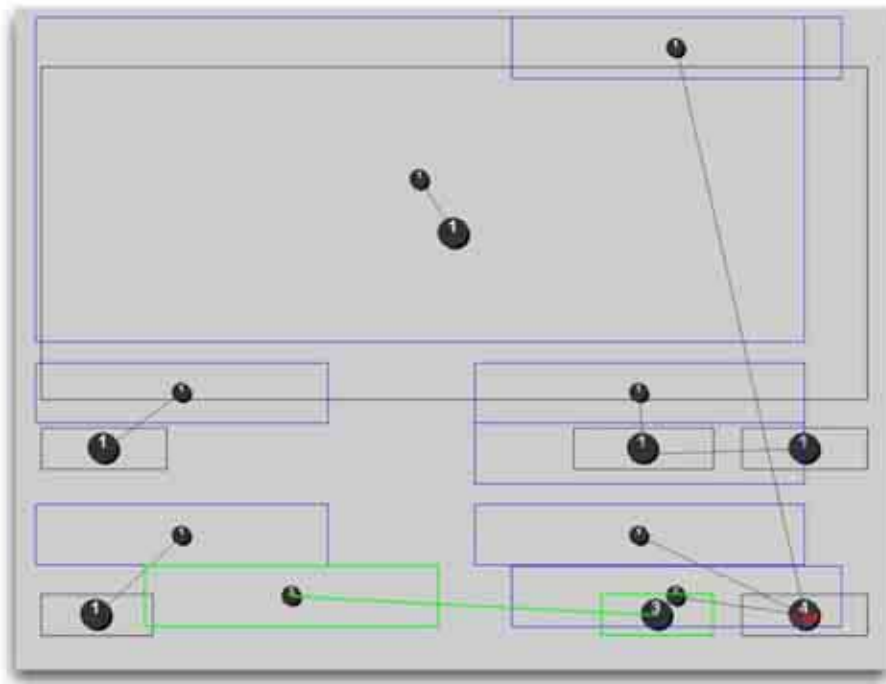


Abbildung 6.38: Die Entwicklungsumgebung gibt Rückmeldung über die Verbesserung der Konsistenz.

rung der Konsistenz nun alle Elemente in dieselbe Gruppe projiziert und die Häufigkeitsverteilung $3 \rightarrow 3$ Elemente anzeigt. Dieses Element wurde somit konsistent positioniert.

Wie dieses Beispiel zeigte, kann durch Übernahme der anwendungsweiten Perspektive mit Hilfe der transformationalen Konsistenzbestimmung ein Bewusstsein für die plattformübergreifend konsistente Darstellung geschaffen werden. Die notwendigen Methoden zur Unterstützung der Entwickler bei der Gestaltung solcher Anwendungen beschränkt sich hierbei nicht auf komplette Neuentwicklungen von Darstellungssystemen (s. Abschnitt 6.1) und modellbasierten Konzepten (s. Abschnitt 4.3.2), sondern kann mittels der transformationalen Konsistenz auch in bestehende Werkzeuge eingebunden werden.

Die hier vorgestellte Anwendung wurde mit *Java* entwickelt. Eine Einbettung in die *Open-source* Entwicklungsumgebung *Eclipse* oder in andere Entwicklungsumgebungen wie *GraphiXML* [LV04] sind möglich. Wünschenswert jedoch ist die Berücksichtigung derartiger Konzepte vor allem in den wesentlichen Entwicklungsumgebungen, wie dem *Visual Studio* von *Microsoft*.

6.2.3.2 Transformation von Elementgruppen

In Abschnitt 5.3 wurden verschiedene Transformationsstrategien beschrieben. Es wurde postuliert, dass sich eine jede Abbildung zwischen Plattformen in einzelne atomare Transformationen zerlegen lässt. Konstruktiv angewendet bedeutet dies, dass die Abbildung einer Darstellung in eine andere durch die Bereitstellung solcher Transformationsstrategien unterstützt werden kann.

Die Bereitstellung von Methoden zur Anwendung von Transformationen bei der Anpassung von Benutzerschnittstellen sollte daher die Konsistenz der Abbildung unterstützen. So wird es ermöglicht, Gruppen von Elementen einfach und schnell auf eine neue Plattform abzubilden. Generalisiert man diese Abbildung dann auf alle ähnlichen Elementgruppen würde dies bedeuten, dass die Abbildung konsistent über die verschiedenen Dialoge der Anwendung erfolgt, die Abbildung also *transformationally consistent* ist.

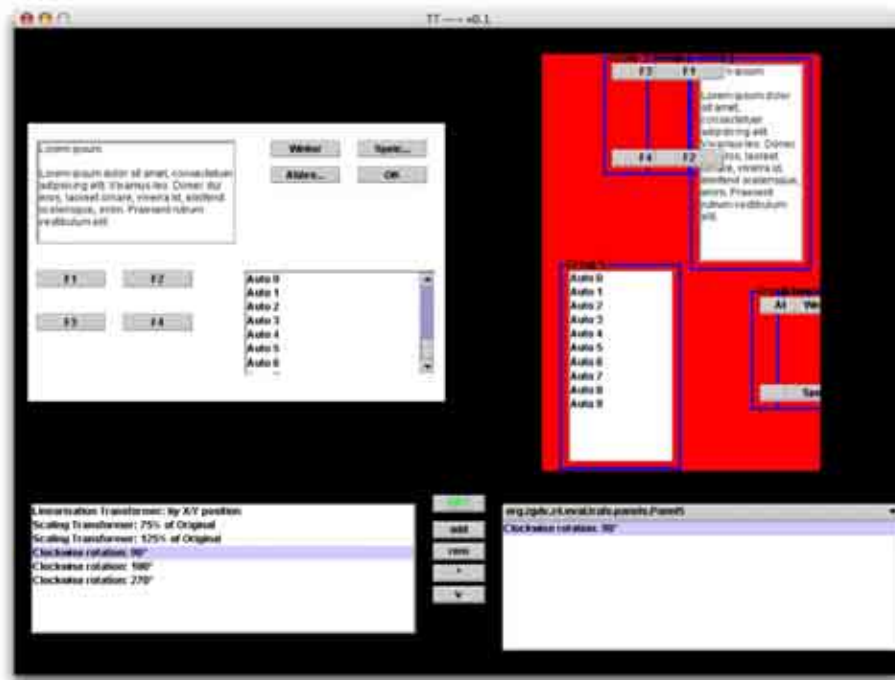


Abbildung 6.40: Die Optimierung des Transformationsprozesses

Algorithmus 1 Vereinfachte Darstellung des Optimierungsalgorithmus

```

1: procedure OPTIMIZE
2:   for  $i = 0$  to  $Screen.height$  do
3:      $Group \leftarrow GroupControls(Controls(i))$ 
4:     if  $Control.x + Control.width > Screen.width$  then
5:        $ShiftLeft()$ 
6:     end if
7:     if  $\sum_{j=0}^n (Control_j.width) > Screen.width$  then
8:        $ScaleWidth()$ 
9:     end if
10:  end for
11: end procedure
12: function GROUPCONTROLS(height)
13:    $Group \leftarrow Controls \forall \{y = Group.y \& width = Group.width \& height = Group.height\}$ 
14:    $\triangleright$  Gruppieren Elemente mit gleichen Koordinaten und Maßen
15: end function
16: function SHIFTLLEFT
17:    $Group.x = 0$   $\triangleright$  Verschiebe Elemente nach links
18: end function
19: function SCALEWIDTH
20:    $Group.width' \leftarrow reduce(Group.width, Screen.width)$   $\triangleright$  Reduziere auf passende Breite
21:   if  $Group.width' > Screen.width$  then
22:      $wrap(Group)$   $\triangleright$  Breche Gruppe um.
23:   end if
24: end function

```

Abbildung 6.41 zeigt ein Beispiel einer Transformation für eine Gruppe von Elementen innerhalb einer Plattform. Der Benutzer selektiert in der visuellen Darstellung eine Gruppe von Elementen und wählt über das Kontextmenü eine Transformationsstrategie, die auf diese Gruppe angewendet

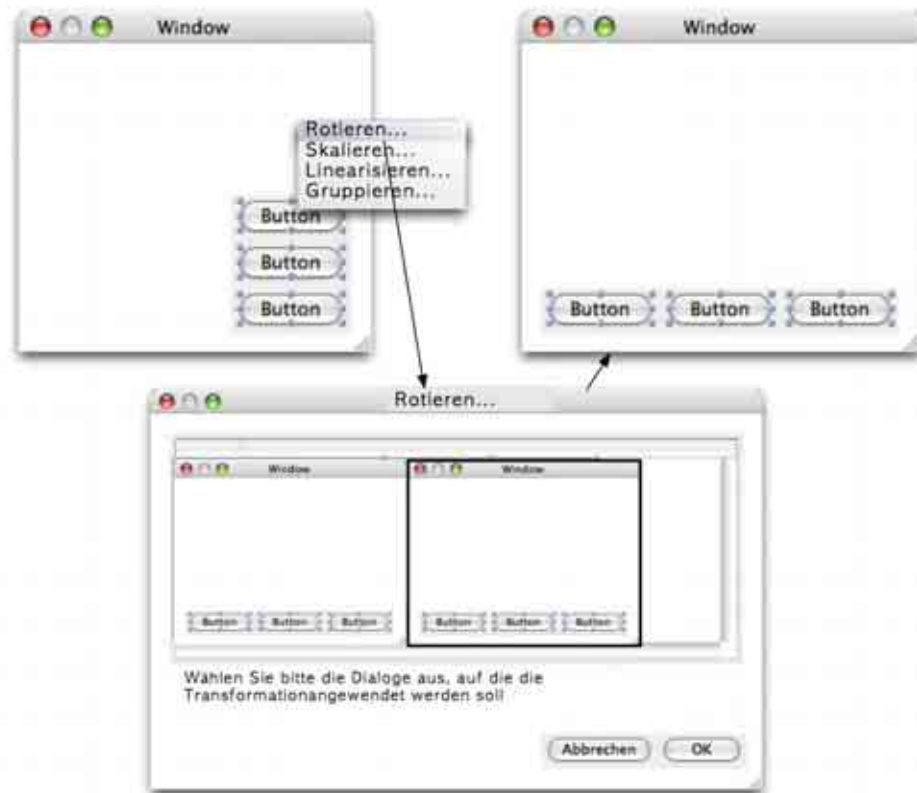


Abbildung 6.41: Beispiel für die Transformation einer Element-Gruppe

werden soll. In dem Beispiel wird eine Rotation gewählt. Ein Dialog erscheint, in dem das Resultat der Transformation für alle Dialoge, die diese Konfiguration enthalten, als Vorschau angezeigt wird.

Analog zu dem Prozess in Abschnitt 6.2.3.1 werden über alle Dialoge Gruppen von Elementen gesucht, die sich bezüglich eines oder mehrerer Kriterien innerhalb einer bestimmten Toleranz gleichen. Entsprechend dem Transformationsparadigma wird diese Transformation dann auf alle ähnlichen Elementgruppen angewandt und in der Vorschau angezeigt.

Der Benutzer kann nun wählen, für welchen Dialog diese Transformation angewendet und welche Dialoge ausgespart werden sollen. Diese Entscheidung hat entsprechende Folgen für die Konsistenz innerhalb der Anwendung und sollte vom Designer mit Bedacht ausgewählt werden.

Abbildung 6.42 zeigt ein Beispiel, wie diese Anwendung alternativ auch bei der Transformation zwischen Plattformen eingesetzt werden kann. Soll ein Dialog für die Desktop Plattform z. B. auf ein Smartphone abgebildet werden, so sind hier insbesondere Anpassungen in der Größe, aber auch bei der Auswahl der Elemente vonnöten. Um in einem automatischen Übersetzungsprozess die Kontrolle durch den Designer zu ermöglichen, kann hier eine geleitete Transformation angewandt werden.

Der Designer selektiert ein Dialogelement und wählt eine Transformation. Es werden alle ähnlichen Elemente in allen Dialogen der Anwendung gesucht, z. B. alle Elemente die an derselben Stelle stehen oder schlicht alle Listenelemente. Der Vorschau-Dialog zeigt welche Dialoge von einer derartigen Anpassung betroffen sind und wie diese aussieht. Ein weiterer Zwischenschritt ermöglicht die Auswahl des Elements, durch welches das ursprüngliche Element ersetzt werden soll. Der Designer kann auch hier wählen, für welchen Dialog die Transformation gelten soll. Durch das Bestätigen der Anpassung wird die Transformation durchgeführt.

Diese Form des *Refactoring* ist eine wichtige Möglichkeit, konsistent Änderungen an bereits implementierten Anwendungen vorzunehmen ohne durch sämtliche Dialoge der Anwendung gehen zu müssen. *Refactoring* ist nach Reiterer [Rei00] eine wichtige Unterstützung *agiler* Entwicklung, bei

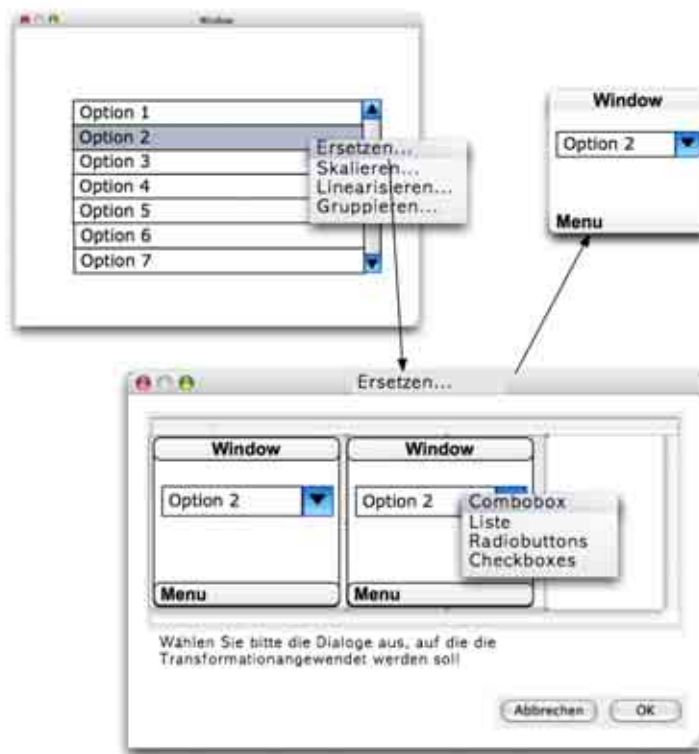


Abbildung 6.42: Ersetzung eines Elements bei der Transformation zwischen Plattformen.

der aufgrund der schnellen Umsetzung nachträgliche Änderungen meist zu umfangreichen, stark im Projekt verteilten Anpassungen kommt.

Bereits in Abschnitt 2.3 wurde auf die Bedeutung von Standards hingewiesen, welche eine Form der normativen Wissensvermittlung in Gestalt von Vorschriften und Richtlinien darstellen. Reiterer [Rei00] verweist weiterhin auf die Verwendung von sogenannten *Style Guides* in denen Gestaltungsvorschriften meist im Rahmen einer Organisation oder der gestalterischen Konsistenz einer Softwareplattform gegeben werden. Nachteil dieser Vorschriften ist ihre weitestgehende Nichtbeachtung durch die Entwickler aufgrund deren mangelnder Integration in den Entwicklungsprozess [MN90, BHW92].

6.2.3.3 Anwendung von Transformationsmustern

Eine der grundlegendsten Formen der Designunterstützung ist die Vermittlung von Designwissen, die Vermittlung von Lösungen, Methoden und Kriterien zur Optimierung der Benutzbarkeit von Oberflächen. Die Bereitstellung von Unterstützungsmechanismen welche unmittelbar in die Entwicklungsumgebungen eingebunden werden können, wie die Beispiele in den vorangegangenen Abschnitten zeigen, ermöglichen eine direkte Unterstützung des Prozesses durch Leitung des Entwicklers.

Eine andere Form der Unterstützung stellt die Bereitstellung von prozessbegleitenden Methoden, wie z. B. dem PROUSE-System [MR02b, MR02a, EMB03] dar. In diesem System werden prozessbegleitend Wissensinhalte aus der Domäne des *Human-Centered-Design (HCD)* angeboten. Methoden, *Best-Practice*, Werkzeuge und Richtlinien werden entlang des HCD-Prozesses angeboten und können im Gegenzug durch den Nutzer bewertet werden.

Eine weitere Möglichkeit der Unterstützung (und denkbar als Teil eines prozessbegleitenden Werkzeugs) besteht in der Bereitstellung von prototypischen Lösungen in ihrem spezifischen Kontext. Beispielhafte Realisierungen von Lösungen für Probleme und für Gestaltungsansätze, die sich in

der Praxis bewährt haben, können oftmals besser aufgenommen (soziales Lernen [OM95]) und effektiver umgesetzt werden als abstrakt verallgemeinerte *Guidelines*. Die Verwendung von Musterlösungen, sogenannten *Patterns* kann daher auch bei der Entwicklung von plattformübergreifenden Benutzerschnittstellen sinnvoll eingesetzt werden

Der Architekt Christopher Alexander [Ale79] prägte den Begriff der *Design Patterns* als er die nicht konkret greifbaren Eigenschaften bestimmter baulicher Lösungen in existierenden Gebäuden, ernannte diese die *Quality Without a Name*, zu beschreiben und wichtiger noch, zu sichern und vermittelbar zu machen versuchte. Die Qualität einer Lösung jedoch, so Alexander, lässt sich meist nur von deren Benutzern, d. h. den Bewohner beschreiben.

“Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution.” [Ale79]

Ein einfacher Weg, solche Muster zu beschreiben wurde notwendig. Alexander *et al.* [AIS⁺77] definierten eine Struktur, wie solche *Design Patterns* beschrieben werden sollten. Er identifizierten deren wesentliche Bestandteilen als: Namen, Bewertung, Abbildung, Anwendungskontext, Problembe-schreibung, Beschreibung der Lösung, schematische Beschreibung und Verweise auf andere Muster.

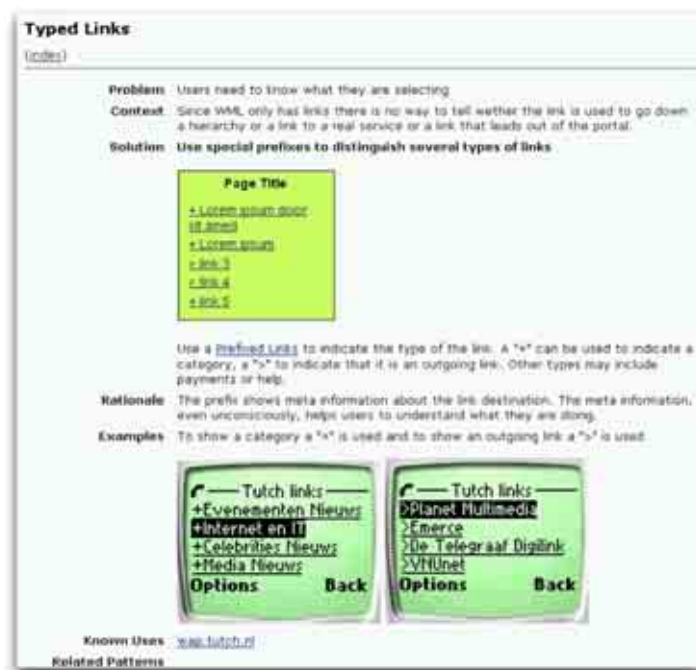


Abbildung 6.43: Beispiel eines HCI-Pattern von Welie.com

Mit der Entwicklung hin zur Objektorientierung und den damit verbundenen formalen Methoden [RJB97] fanden *Patterns* vermehrt Anwendung im Software-Design [GHJV94, Bor01]. Insbesondere in der Forschung und Entwicklung der Mensch-Maschine Schnittstelle, wo die Komplexität der Wechselwirkungen die der formalisierbaren Algorithmen in der Softwareentwicklung weit übersteigt, wurde die Möglichkeit der *HCI-Patterns* schnell erkannt [WV03, DFAM02, Nor88]. Die Anwendung von *Patterns* zur Beschreibung von Gestaltungsmöglichkeiten bei der Entwicklung hat zu einer großen Zahl von sogenannten *Pattern-Bibliotheken* geführt [App00, CL96, Fin00, Wel05, WMM02]. Ein Beispiel für ein *Pattern* für Mobilgeräte aus der Galerie von van Welie [Wel05] kann in Abbildung 6.43 betrachtet werden.

Javahey *et al.* [JSES04] diskutieren die Anwendung von *HCI-Patterns* bei der Anpassung von plattformübergreifenden Anwendungen anhand des Beispiels einer webbasierten Anwendung. Javahey *et al.* gehen hierbei so vor, dass sie zunächst die Darstellung in eine Vielzahl von *HCI-Patterns*, wie dem *Bread Crumbs Pattern*, *Portal Pattern*, etc., zerlegen. Für jede Zielpattform existiert meist

schon eine Realisierung dieses *Patterns*, so dass dies direkt übersetzt werden kann. Abbildung 6.44 zeigt diesen Übersetzungsprozess in einer schematischen Übersicht.

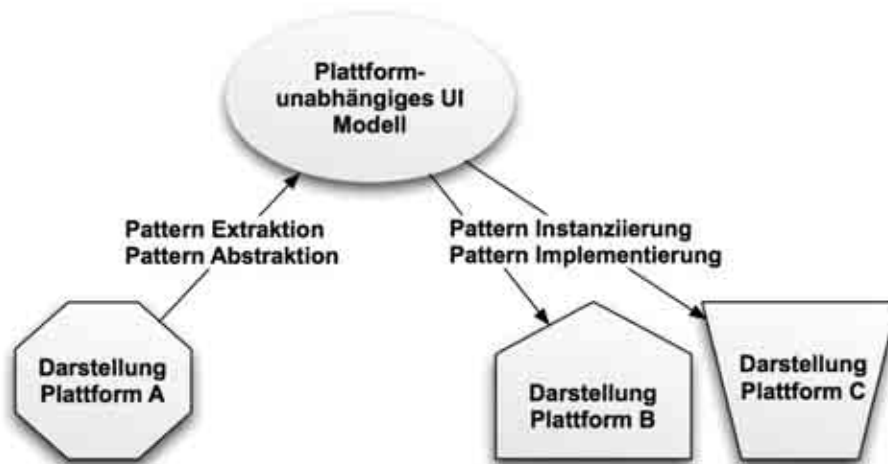


Abbildung 6.44: Patternunterstützte Übersetzung von Benutzerschnittstellen nach [JSES04].

Wie Javahery *et al.* vorschlagen, sollen in einem Transformationsprozess die relevanten *Patterns* (der Begriff wird hier soweit gefasst, dass quasi alle Elemente darunter fallen) extrahiert und in eine plattformunabhängiges Modell der Benutzerschnittstelle abstrahiert werden. Diese Abstraktion kann dann mittels konkreter plattformspezifischer Instanzen der einzelnen *Pattern* konkretisiert und implementiert werden.

Wenngleich der Transformationsprozess von Javahery *et al.* derzeit nur konzeptionell existiert und sicherlich problematisch in der Umsetzung ist, so scheint die Anwendung von *Patterns* für die plattformübergreifende Gestaltung von Benutzerschnittstellen als Unterstützungsmethode geeignet. Der folgende Abschnitt beschreibt exemplarisch drei *Design-Patterns* zur Anpassung von Benutzerschnittstellen auf Mobilgeräte mit eingeschränkter Bildschirmfläche.

- *Zoomable Interfaces (DateLens, FishEye-Menus)*
- *Mobile Liquid Scatter Space (iSpace)*
- *Scrollable Toolbars*

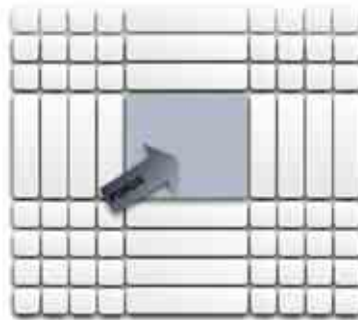
Derartige *Patterns* könnten im Rahmen von *Transformation-Patterns* in Galerien gesammelt, und dem Benutzer als Informationsquelle zur Verfügung gestellt werden. Anhand der Problemstellung und der Zielpattformen können derartige *Patterns* durchsucht und dem Designer problembezogen präsentiert werden.

Zoomable Interfaces

Problem: Eine komplexe tabellarische Darstellung soll auf einem Gerät mit geringer Bildschirmgröße dargestellt werden. Die Fläche ist zu gering um alle Informationen zugleich darzustellen. Der Benutzer soll dennoch möglichst weitestgehend die Übersicht über alle Daten zur gleichen Zeit erhalten und dennoch Detailinformationen anzeigen können.

Kontext: Die Daten, welche dargestellt werden, besitzen eine regelmäßige Struktur. Die Darstellung auf dem Zielgerät ist hochauflösend und die Eingabe erfolgt über ein Point & Click Gerät (Maus oder Stift) direkt-manipulativ. Eine farbige Darstellung unterstützt diese Darstellungsweise.

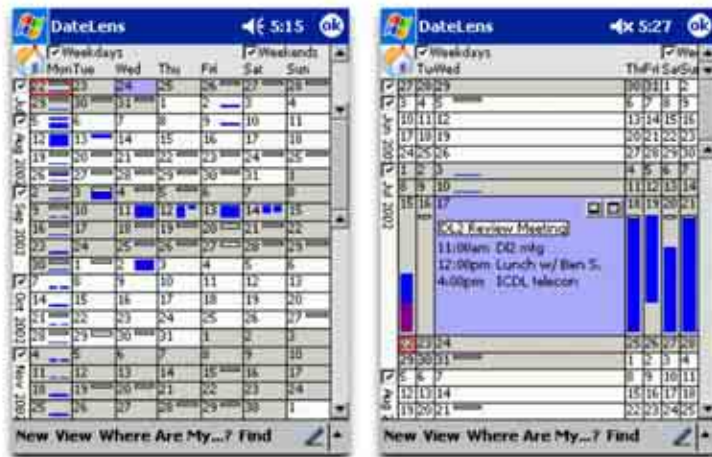
Lösung: *Nutze eine skalierbare Darstellung, bei der die augenblicklich wichtigen Elemente (Fokus) vergrößert werden.*



Wenn der Benutzer mit dem Interaktionsgerät über die Darstellung geht, erscheint der Ausschnitt der Informationen, der sich im Fokus befindet Größer als die anderen Bereiche. Der Übergang der Größe erfolgt fließend.

Hintergrund: Die gleichzeitige Hervorhebung der wesentlichen Informationen und die Beibehaltung des räumlichen Kontext durch einen fließenden Übergang und die gestauchte Darstellung des Umfeldes ermöglicht die räumliche Orientierung.

Beispiele: Dieses Muster wird von dem *FishEye Menu* [Bed00] und der *DateLens* Kalendervisualisierung [BCCR04] umgesetzt.



Ziel-Plattformen: • Desktop PC (eingeschränkt sinnvoll)

- Pocket PC
- Smartphone

Verwandte Patterns: • Liquid Scatter Space

Liquid Scatter Space

Problem: Eine große Menge von Datenpunkten oder Ressourcen soll nach verschiedenen Kriterien zu ordnen und zu durchsuchen sein, Die Darstellungsfläche dazu ist eingeschränkt und textuelle Eingabe sollte aufgrund einer stiftbasierten Eingabe nicht nötig sein.

Kontext: Die Daten, welche dargestellt werden, werden mit den Metadaten versehen, nach denen sortiert wird. Die Darstellung auf dem Zielgerät ist hochauflösend und die Eingabe erfolgt über ein Point & Click Gerät (Maus oder Stift) direkt-manipulativ. Eine farbige Darstellung unterstützt diese Darstellungsweise. Das Gerät benötigt eine ausreichende Rechenleistung.

Lösung: Ermögliche es dem Nutzer in den Daten zu wählen.



Durch den Zeiger kann der Benutzer die Elemente auseinander schieben wie mit zwei gleichpoligen Magneten. Auf diese Weise können Cluster durch den Zeiger entzerrt und durchsucht werden. Die Abstoßung vom Zeiger erfolgt in einem kreisförmigen Bereich. Geht der Fokus fort fallen die Elemente wieder in die ursprüngliche Lage zurück.

Hintergrund: Durch die direkt-manipulative Interaktion mit den Daten entsteht das Gefühl für den Benutzer er würde die Daten durchwühlen. Die möglich zusätzliche Dimension der Druckstärke kann die Größe des Abstoßungskreises beeinflussen und gibt dadurch einen noch direkteren Eindruck.

Beispiele: Dieses Muster wird von dem iSpace Liquid Browser [WB04, WHB03] und in einer speziellen Anwendung als Datei-Browser in [Wal05] umgesetzt.



Ziel-Plattformen: • Large Displays

- Desktop PC
- Pocket PC
- Smartphone

Verwandte Patterns: • Zoomable Interfaces

Scrollable Toolbar

Problem: Eine Symbolleiste mit vielen Elementen soll auf einem Gerät mit Platz für nur wenige Elemente dargestellt werden.

Kontext: Auf dem Gerät können n Elemente in einer Symbolleiste dargestellt werden. Es müssen jedoch m Elemente dargestellt werden, wobei gilt, dass $n < m < (2 \times n)$. Das Zielgerät kann mit einem Zeigegerät bedient werden.

Lösung: *Erzeuge eine scrollbare Symbolleiste.*



Der Benutzer kann durch Anwahl eines Steuerelements den Bereich der sichtbaren Elemente nach rechts oder links verschieben und so einen anderen Bereich an Elementen anschauen. Zusätzlich kann eine Darstellung der momentanen Position im Gesamtbereich (Scrollleiste) angezeigt werden und die Position aktiv darüber manipuliert werden.

Hintergrund: Die Scrollleisten-Metapher ist den Benutzern von Windows-Systemen eine geläufige Metapher und sie ermöglicht den einfachen Zugriff auf eine Menge von Elementen, welche größer ist als die darstellbare Fläche.

Beispiele: Dieses Muster wird in Abschnitt 7.2.2 beschrieben und umgesetzt .



Ziel-Plattformen:

- Pocket PC
- Smartphone

Verwandte Patterns:

- Switchable Toolbars

6.3 Zusammenfassung

In diesem Abschnitt wurde die Umsetzung der Konzepte der plattformübergreifenden Entwicklung von Benutzerschnittstellen aus Abschnitt 5 beschrieben. Diese Umsetzung gliederte sich in zwei Schwerpunkte. Zunächst wurde die Realisierung eines Darstellungssystems und der *bottom-up* Entwicklung von der Anwendung kommend betrachtet. Im zweiten Teil wurden dann Methoden zu Unterstützung der *top-down* Entwicklung in Form graphisch-interaktiver Design-Prozesse entwickelt.

Wie in der Anforderungsanalyse (s. Abschnitt 2.4) beschrieben, ist es notwendig, neue Entwicklungsmethoden in die existierenden Prozesse einzubinden und den Lernaufwand für den Entwickler zu reduzieren, um so die Akzeptanz zu erhöhen und die Schwelle zur Einführung zu verringern. Dies war das vorrangige Ziel der *bottom-up* Methoden. Die Unterstützung der heute üblichen toolkitbasierten Programmierung als auch die strikte Trennung von Datenmodell und Darstellung ermöglichen die Vereinfachung der Entwicklung. Zwei *Programmierschnittstellen*, von denen beide in verschiedenem Maße das zugrundeliegende Darstellungs- und Datenmodell kapseln, sollen es ermöglichen ohne tiefere Kenntnisse der inneren Prozesse, abstrakte Oberflächen zu realisieren. Die Abspaltung der Darstellung vom Datenmodell und die sequentielle Anpassung stellen eine mögliche Lösung des *Mapping-Problems* dar. Das System wurde im Rahmen dieser Arbeit exemplarisch in der *Microsoft .Net Plattform* für PC, Pocket PC und Smartphone umgesetzt.

Aus der Perspektive des Designers interaktiver Systeme wurde gezeigt, dass insbesondere in der plattformübergreifenden Entwicklung die Bewältigung des *Optimierungs- vs. Konsistenzproblems* von Bedeutung ist. Hierzu sind neue Methoden der Unterstützung des Designprozesses notwendig. Ein Schwerpunkt dieses Abschnittes war es daher, Methoden vorzustellen, wie die Bewertung der plattforminternen, sowie plattformübergreifenden Konsistenz unterstützend in den Designprozess eingebunden werden kann. Insbesondere die *anwendungsweite Perspektive* und die dadurch mögliche anwendungsweite Evaluation spielen hierbei eine zentrale Rolle. Die Konsistenzmessung stellt eine direkte Anwendung des Transformationsparadigmas und des daraus abgeleiteten transformationalen Konsistenzmaßes als zentralem Beitrag dieser Arbeit dar.

Neben der hier beschriebenen Umsetzung, welche als *proof-of-concept* die technische Realisierbarkeit der Konzepte belegen und Beispiele für deren konkrete Anwendung in der Entwicklung plattformübergreifender Benutzerschnittstellen bieten soll, werden im nächsten Abschnitt empirische Belege dieser Konzepte und ihrer Realisierung präsentiert.

Wie in diesem Abschnitt gezeigt werden konnte, eignen sich die Konzepte, welche im Rahmen dieser Arbeit in Abschnitt 5 auf Basis der wissenschaftlichen Grundlagen und unter Berücksichtigung aktueller Methoden entwickelt wurden, als Basis für konkrete Entwicklungssysteme. Dies wurde mit der Umsetzung eines exemplarischen Darstellungssystems, sowie durch die Beschreibung und teilweise Umsetzung von entwicklungsunterstützenden Methoden basierend auf dem transformationalen Konsistenzmaß nachgewiesen.

7 Systemvalidierung

Die Relevanz und Validität der in dieser Arbeit vorgeschlagenen Konzepte wird in diesem Abschnitt anhand empirischer Methoden überprüft. Diese Abschnitt gliedert sich entsprechend der thematischen Schwerpunkte und wesentlichen Beiträge dieser Arbeit in drei Teile: Validierung der Darstellungssystems, der Transformationshypothese und schließlich des Konsistenzmaßes.

Validierung des Darstellungssystems: Die Entwicklung plattformübergreifender User Interfaces soll entsprechend der Anforderungen kompatibel zu etablierten Entwicklungsansätzen und möglichst wenig komplex sein. Das Darstellungssystem, wie es in Abschnitt 6.1 beschrieben wurde, bietet zum einen verschiedene toolkit-basierte Zugriffe welche die Entwicklung plattformübergreifender User Interfaces vereinfachen und komplexe Interne Zusammenhänge auf verschiedenen Abstraktionsebenen kapseln. Auf diese Weise können schnell und flexibel erste funktionale Versionen einer Anwendung realisiert werden und in der Folge iterativ *bottom-up* (über die APIs) oder *top-down* (über einen graphischen Editor) verfeinert werden. Dieses System eignet sich damit als Grundlage für verschiedene Entwicklungsmethodiken wie den klassischen prozessorientierten aber auch den modernen *agilen* Ansätzen. Der erste Teil der Validierung konzentriert sich auf die Umsetzung dieses Darstellungssystems in einer frühen Form als Teil eines nationalen Forschungsprojektes.

Validierung der Transformationshypothese: Eine wesentliche Hypothese dieser Arbeit ist die Transformationshypothese. Diese kognitionspsychologisch motivierte Hypothese geht davon aus, dass bei der Benutzung derselben Anwendung auf zwei Plattformen dies vom Benutzer als Transformation der Anwendung wahrgenommen wird. Er versucht daher, die Versionen ineinander zu überführen. Plattformübergreifender Design kann daher besser gestaltet sein, wenn es dem Benutzer erlaubt, diese mentale Transformation einfach durchzuführen. Diese wird z. B. ermöglicht, indem die elementaren Basistransformationen in regelmäßiger Weise angewendet werden. Voraussetzung für die Gültigkeit dieser Hypothese ist zunächst, dass der Benutzer tatsächlich eine derartige Transformation vollzieht. Anhand des bekannten Phänomens der mentalen Rotation diese Annahme überprüft. Weiterhin wird eine regelbasierte Transformation unter Verwendung eines Transformationsmusters aus Abschnitt 6.2.3.2 auf ihre Wirkung bezüglich der Benutzbarkeit mit einer händisch angepassten, kommerziell verfügbaren Variante verglichen.

Validierung des Konsistenzmaßes: Aufbauend auf der Transformationshypothese wurde in Abschnitt 5.4 ein Maß der Konsistenz entwickelt, welches geeignet ist die Regelmäßigkeit der Beziehung verschiedener Darstellungsaspekte einer Benutzungsschnittstelle zu operationalisieren. Das Maß kann sowohl zur Evaluierung der Konsistenz von Anwendungen auf einer Plattform als auch für die Messung der plattformübergreifenden Konsistenz angewandt werden. Dieses transformationale Konsistenzmaß kann, wie in den Abschnitten 6.2.2 und 6.2.3 beschrieben zur Unterstützung des Designprozesses genutzt werden und stellt einen zentralen Beitrag dieser Arbeit dar. Der Letzte Teil der Validierung beschäftigt sich daher mit der empirischen Überprüfung dieses Konsistenzmaßes.

7.1 Validierung des Darstellungssystems

Die Umsetzung des in Abschnitt 6.1 vorgestellten Darstellungssystems wurde als Teil eines Forschungsprojektes erstmals realisiert und auf seine Benutzbarkeit evaluiert. Der Fokus dieser Umsetzung lag hierbei im wesentlichen auf der Darstellung plattformunabhängiger Benutzerschnittstellen in multimodalen Systemen und die Anpassung der Standarddarstellung auf die Bedürfnisse des individuellen Benutzers.

Wesentliche Modifikationen bezüglich des oben beschriebenen Systems betrafen die Einbettung in eine *verteilte agentenbasierte Architektur*, die Entwicklung einer *multimodalen Darstellungspipeline* und der konkreten *Umsetzung von Anpassungsstrategien* insbesondere für an die Anforderungen an behinderte Benutzer.

Die Evaluierung des Darstellungssystems gliedert sich aus diesem Grund in drei Teile. Zunächst wird eine kurze Einführung in die Motivation und den Anwendungskontext des Projektes gegeben. Diese werden insbesondere in Beziehung auf die, in der Motivation (Abschnitt 1.1.2) und den Anforderungen (s. Abschnitt 2.3) angeführten Zielen dieser Arbeit eingeordnet. Weiterhin werden konkrete Aspekte der technischen Umsetzung, soweit diese die Schilderungen in Abschnitt 6.1 ergänzen, erläutert.

Die Evaluierung des Darstellungssystems bezieht sich im wesentlichen auf die Bewertung der Bedienbarkeit durch Benutzer mit besonderen Anforderungen. Es wird hier insbesondere auf die Nützlichkeit des in dieser Arbeit verfolgten Ansatzes im Kontext der Schaffung universell nutzbarer (*Design for All*) Benutzerschnittstellen eingegangen.

7.1.1 Das Leitprojekt EMBASSI

Das *EMBASSI*-Projekt war ein vom deutschen Bundesministerium für Bildung und Forschung gefördertes Leitprojekt zur Erforschung neuer Wege in der *Mensch-Technik-Interaktion (MTI)*. Ziel des Projektes, war es, „ein ganzheitliches Assistenzkonzept zu entwickeln, das den Nutzer bei der Bedienung von Alltagstechnologie optimal unterstützt“ [EMB03]. Das Projekt gliederte sich in drei Anwendungsbereiche für die diese Konzepte exemplarisch entwickelt werden sollten [WNPW02]: Privathaushalt, Automobiles und Öffentliches Umfeld.

Das in Abschnitt 6.1 beschriebene Darstellungssystem wurde im Anwendungsszenario *Öffentliches Umfeld* mit der konkreten Anwendung „Unterstützung der Interaktion von Behinderten mit Terminalsystemen durch persönliche mobile Assistenzgeräte“ [MR01, Ric01, RH04a, RH04b] umgesetzt. Die hier vorgestellten Konzepte betreffen in erster Linie den Einsatz einer abstrakten Beschreibungssprache für plattformunabhängige und multimodale User Interfaces, deren Entwicklung und Umsetzung sowie die Umsetzung einer generischen Darstellungsplattform für diese. Im EU-Projekt *I2HOME* [ARB06, ARZ⁺05] dienten Teile dieses Ansatzes als Grundlage bei der Beantragung (Projekt Homepage: <http://www.i2home.org>).

Um die im *EMBASSI* Anwendungsbereich *Terminalsysteme* adressierten Ziele zu verdeutlichen, wurden verschiedene Anwendungsszenarien definiert, die als Richtschnur und als Umsetzungsziele für eine Systemvalidierung dienten. Zunächst soll an dieser Stelle eines dieser Szenarien exemplarisch zur Verdeutlichung der Ziele erläutert werden.

In dem hier vorgestellten Szenario verwendet eine motorisch eingeschränkte Person ihren persönlichen mobilen Assistent zu Hause um ihren nächsten Einkauf zu planen. Die Eingaben können über eine spezielle Steuerung über das Kinn erledigt werden (s. Abbildung 7.1). Weiterhin steht eine spezielle Lautsteuerung zur Verfügung, welche die Eingabe über vokalische Äußerungen ermöglicht und keine klare Artikulation von Worten erfordert, da dies der betroffenen Person nicht möglich ist. Ein Einkaufsassistent auf ihrem Mobilgerät erlaubt es ihr in Ruhe zu Hause die Einkaufsliste zu vervollständigen. Dieser Assistent bietet Filterassistenz, indem er die bevorzugten Artikel aus früheren Einkäufen anbietet. Ebenfalls stehen komplette Einkaufslisten zur Auswahl. Angebotsassistenz ermöglicht weiterhin die Sichtung der verfügbaren Artikel aus früheren Einkäufen. Neue Artikel können in Freitext eingegeben werden. Ein Wortergänzungsprogramm, welches anhand der aus der Dialogbeschreibung verfügbaren Kontextinformationen gezielt Vorschläge (Beraterassistenz) machen kann unterstützt die Eingabe.

Nachdem die Person die Liste komplettiert hat, speichert sie diese und verlässt das Haus. Sie begibt sich mit ihrem Rollstuhl in die Stadt. Nähert sie sich einem Einkaufsterminal informiert sie der mobile Assistent darüber und erinnert sie an die Liste. *EMBASSI*-Terminals publizieren die verfügbaren Dienste über ein drahtloses Netzwerk an alle mobilen Assistenten in einem bestimmten Umkreis. Die Person beschließt den Dienst zu nutzen und nähert sich dem Terminal und startet eine *EMBASSI*-Sitzung. Der reguläre Bildschirm des Terminals wird gesperrt, Eingaben sind nicht sichtbar da dies in dem Benutzerprofil als Präferenz gespeichert ist. Der Benutzerassistent sendet nun die Liste an den Terminal und gleicht die Liste mit dem Angebot ab. Unklare oder nicht verfügbare Artikel werden interaktiv mit dem Benutzer abgeglichen, wenn dieser es wünscht. Die komplexe Eingabe auf dem Terminal wird durch den persönlichen Assistenten auf wenige Interaktionsschritte



Abbildung 7.1: Eine visuell eingeschränkte Person bei der Interaktion mit der graphischen Oberfläche mit Hilfe des taktilen Displays (rechte Hand) und Braille-Zeile.

reduziert. Interaktionen kann der Benutzer entsprechend seinen Präferenzen und Anforderungen mit den Mitteln des Mobilgerätes erledigen.

Dieses Szenario wurde im Rahmen des Projektes realisiert. Der Einsatz des Systems in diesem Szenario wurde dann empirisch von Benutzern mit Behinderungen getestet.

7.1.1.1 Einsatz in einer verteilten Agentenplattform

Im Rahmen des Projektes wurde ein Dialogkonzept für multimodale Interaktion entwickelt [HK01]. Die in Abbildung 7.2 dargestellte konzeptuelle Architektur wurde für die verschiedenen Anwendungsbereiche unabhängig umgesetzt.

Das plattformübergreifende Darstellungssystem welches in dieser Arbeit vorgestellt wird, konnte in einer frühen Version in diese Architektur eingebettet und erfolgreich eingesetzt werden. Das Darstellungssystem musste in dieser Architektur über verschiedenen Ebenen und somit über verschiedene Komponenten verteilt umgesetzt werden. Sowohl der Terminal als auch spezifische Assistenz-Agenten-Module konnten ein User Interface publizieren, welches dann in einem Dialogmanager verwaltet und über den Polymodalen In/Output (PMIO) an die plattform- und modalitätsspezifischen Darstellungsmodule verteilt und dargestellt werden.

Alle Softwarekomponenten, die innerhalb dieses Systems eingesetzt wurden, wurden auf Basis von Agententechnologie realisiert. Agenten sind unabhängig agierende Softwarekomponenten, die über eine Kommunikationsschicht (TCP/IP) und ein Kommunikationsprotokoll (*Knowledge Manipulation and Query Language (KQML)* [LF97]) interagieren. Innerhalb der KQML-Nachrichten werden dann die eigentlichen Nutzdaten, wie z.B. die Beschreibung der Dialoge versendet. Die Konfiguration eines Gesamtsystems steht bei der Nutzung von Agenten meist nicht zur Designzeit fest. Komponenten melden sich über einen Kommunikations und Brokering Server an und können so Nach-

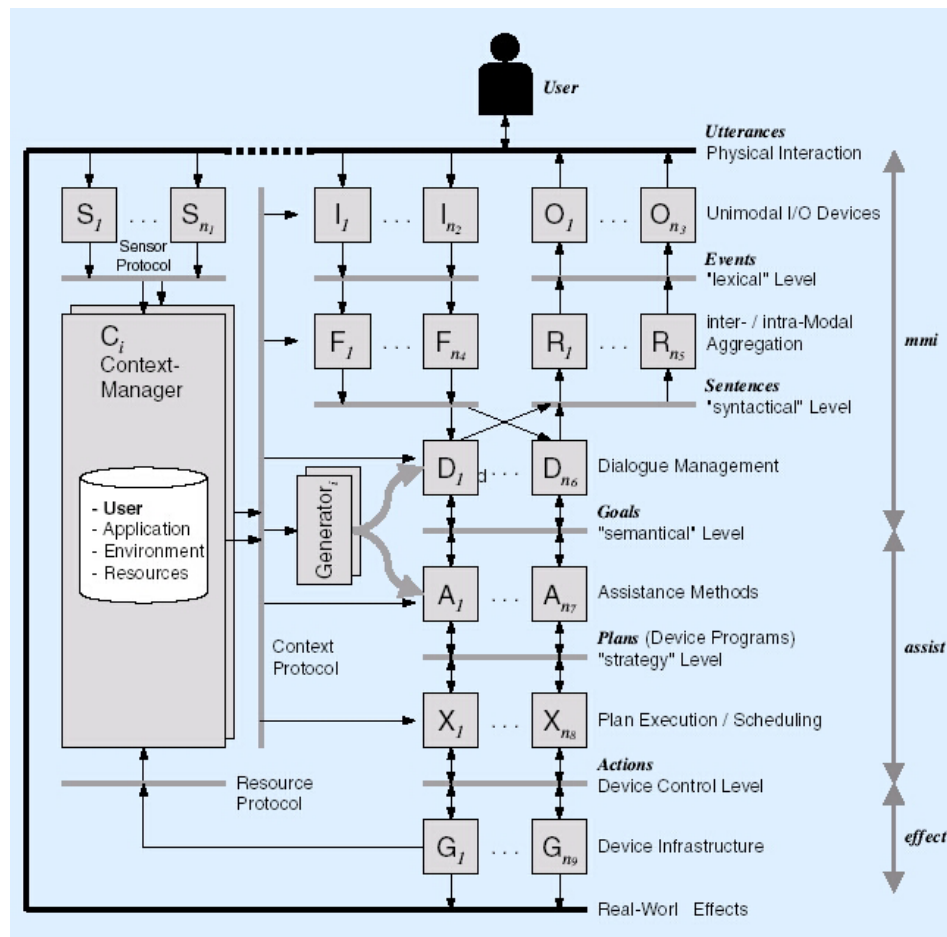


Abbildung 7.2: Multimodales Dialogkonzept nach [HK01]

richten austauschen und Dienste veröffentlichen und nutzen. Im Rahmen des Projektes wurde eine dynamische Agentenplattform basierend auf dem Soda-Pop Framework realisiert [FH02, HKR03].

Clients Innerhalb des EMBASSI Systems, dienen mobile Endgeräte in erster Linie als persönliche mobile Assistenten (*Clients*), welche die Interaktion mit technischen Systemen, hier insbesondere öffentlichen Kiosksystemen, unterstützen. Die mobilen Endgeräte fungieren als Interaktionshilfe mit für auf den Benutzer angepasste Ein- und Ausgabeelemente, als Plattform für personalisierte Assistenzsysteme, sowie als Medium persönliche Informationen, die in der Interaktion zur dynamischen Konfiguration von Diensten genutzt werden kann ohne an zentraler Stelle vorgehalten werden zu müssen.

Aus diesen Gründen wurde im Rahmen des EMBASSI Projektes eine zusätzliche „unsichtbare“ Schnittstelle zu solchen Terminalsystemen geschaffen, welche es den Benutzern ermöglicht Terminal mittels ihres persönlichen mobilen Assistenten zu benutzen. Diese Geräte können private Mobilgeräte sein, welche speziell für die Bedürfnisse ihres Benutzers angepasst wurden. So konnte ein Sehbehinderter seinen Assistenten mit Sprachausgabe und Braille-Zeile ausrüsten, während ein bewegungseingeschränkter Nutzer spezielle Eingabegeräte wie eine Ein-Tasten-Maus nutzen kann. Ein EMBASSI-Terminal liefert nun eine Beschreibung seiner Dienste, bzw. der Benutzungsoberfläche in einer abstrakten Form, so dass diese dann auf dem Mobilten Assistenten in der gewünschten Ausgabeform dargestellt werden konnte. Auf diese Weise wurde ein zusätzlicher Interaktionskanal in Ergänzung zu den üblichen Interaktionsmittel des Terminals genutzt werden.

Zusätzlich zu den angepassten Interaktionsmitteln, welche der mobile Assistent zur Verfügung stellte, konnten personenspezifische Informationen genutzt werden um die Interaktion anzupassen. So

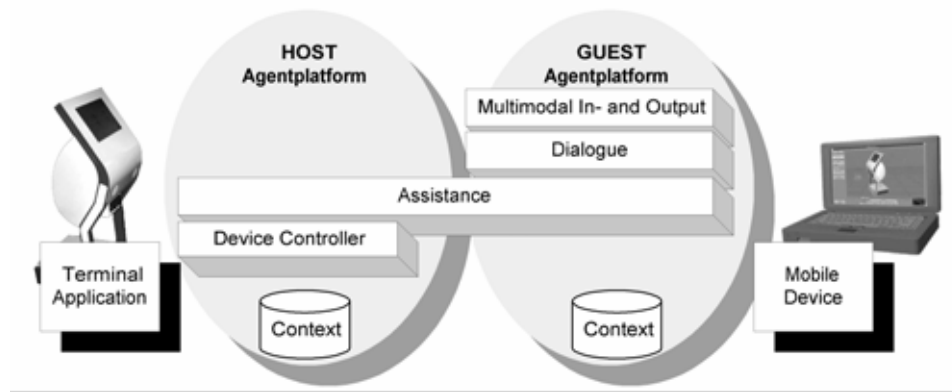


Abbildung 7.3: Übersicht über die Architektur des EMBASSI-Terminalsystems Rahmenmodells.

konnte zum einen das Interface angepasst werden auf die Präferenzen des Benutzers. Zum anderen war aber auch die Verwendung von Profilen zur Erleichterung und Verkürzung von Interaktionen mit Terminalsdiensten möglich.

Terminals Um die Kommunikation mit verschiedenen Terminalsystemen realisieren zu können, wurde stellvertretend für einen Terminal eine Agentenkomponente entwickelt, welche ein Protokoll zur Gerätesteuerung umsetzte. Damit kann die Funktionalität eines Terminals auf sehr allgemeine Weise generisch abgebildet werden und dem System verfügbar gemacht werden. Die Dialoge des Terminals wurden in einer XML-basierten Beschreibungssprache, zunächst in UIML (s. Abschnitt 4.3.1), später in einer Beschreibungssprache basierend auf W3C XForms beschrieben.

Diese Beschreibungssprache entspricht im wesentlichen dem W3C Standard XForms und wurde in eine sog. Host Language eingebettet (s. Abschnitt 6.1). Der statische Teil der Beschreibungen dient als Rahmen für die dynamischen Inhalte, welche aus dem System in Echtzeit abgerufen wurden. So konnte die Schnittstelle zwischen dem Legacy-System (z. B. dem Terminal) und der EMBASSI-Plattform auf ein Minimum reduziert werden. Dies war vor allem durch die Tatsache motiviert, dass auf diese Weise auch existierende Systeme schnell in das EMBASSI-System eingebunden werden sollten. Zwei technische Lösungen wurden hierbei umgesetzt, eine auf XML-Technologien basierende, in der die Systemdaten und die Vorlagen mittels XML Stylesheet Transformationen zu einer User Interfacebeschreibung transformiert wurden und eine zweite, welche über eine API-basierte Programmierung der Oberflächen gestaltet wurde. Bezüglich des Laufzeitverhaltens und der Flexibilität erwies sich der zweite Ansatz als vorteilhaft, so dass dieser Ansatz auch weiterverfolgt wurde.

Ein Terminal versendet über einen Broadcast Nachrichten zu dessen Kennung, Diensten und Netzwerkadresse in ein lokales Funknetzwerk. Mobilgeräte, die diesen Dienst unterstützen, können so ein Terminal erkennen und dessen Dienste anfragen und so eine Verbindung zwischen den Agentenplattformen aufbauen. Da der Terminal als Agent in die Agentenplattform eingebunden wurde, konnte dieser so auch seine Dienste innerhalb der Plattform publizieren und andere Dienste wiederum nutzen. Ein hierarchisches System zur dynamischen Verknüpfung verschiedener Agentenplattformen wurde in diesem Rahmen entwickelt.

Agentenplattform Wie Abbildung 7.3 zeigt, laufen sowohl auf dem Terminal als auch auf dem Mobilgerät jeweils unabhängige Agentenplattformen. Vergleicht man den Aufbau mit der EMBASSI-Referenzarchitektur, welche in Abbildung 7.2 beschrieben ist, sieht man dass die Geräteschicht hier auf der Seite des Terminals gelagert ist, dieser wird über eine Geräte-Controller Komponente, welche als Agent implementiert ist, angebunden. Zusätzliche ressourcenaufwändige Assistenzkomponenten können ebenfalls auf der mächtigeren Terminalplattform laufen. Auf der Seite des Mobilgerätes hingegen werden die höheren Schichten der Referenzarchitektur, wie die Dialogkontrolle und die Darstellung realisiert. Dort kann auch persönliche Assistenz, wie z.B. ein Einkaufsassistent gelagert sein. Eine mobile Plattform kann sich nun an eine Terminalplattform anbinden und erhält somit Zugang zu allen öffentlichen Diensten dieser Plattform.

Die abstrakten User Interface Beschreibungen, die auf der Terminalplattform vom Geräteassistenten versendet werden, werden vom Dialogmanager auf dem Mobilgerät empfangen und dort interpretiert. Der Dialogmanager übernimmt in diesem System in erster Linie die Aufgabe verfügbare Funktionen dem Benutzer zugänglich zu machen. Daher werden hier dynamisch Dienste des Mobilgerätes und Terminals in eine Benutzerschnittstelle verbunden. Unter Umständen müssen so mehrere Dialoge parallel verwaltet werden. Die Beschreibung wird dann in eine Multimodale Darstellungspipeline übergeben, die dann die verfügbaren Ein- und Ausgabekomponenten verwaltet und Inhalte auf diese verteilt, bzw. Eingaben sammelt und integriert. Die Multimodale Ein- und Ausgabe-Pipeline wurde so konzipiert, dass verschiedene Prozessschritte hintereinander geschaltet werden können. Komponenten können durch Implementierung eines Interfaces nachträglich eingefügt werden.

Multimodale Interaktion Die Ein- und Ausgaben können im EMBASSI-System multimodal erfolgen, dies bedeutet, dass verschiedenen Modalitäten (Graphik, Sprache, Haptik, etc.) parallel genutzt werden können. Aus diesem Grund muss die Interfacebeschreibung der Terminals amodal veröffentlicht werden. Diese Beschreibung wird zu dem mobilen Endgerät übertragen und dort dargestellt. Der Benutzer kann dort Eingaben machen, die dann in Kommandos an die Anwendung übersetzt werden. Die Beschreibungssprache, die zu diesem Zweck entwickelt wurde, hat folgende Eigenschaften: sie hat ein geringes Datenvolumen, so dass sie schnell über Netzwerke mit niedriger Bandbreite übertragen werden kann, sie ist adaptierbar und plattformunabhängig, so dass sie auf jedem Endgerät dargestellt werden kann, und zusätzliche Informationen, wie semantische Annotationen zu der Bedeutung von Komponenten sind vorhanden und können so Interaktionsassistenten ermöglichen.

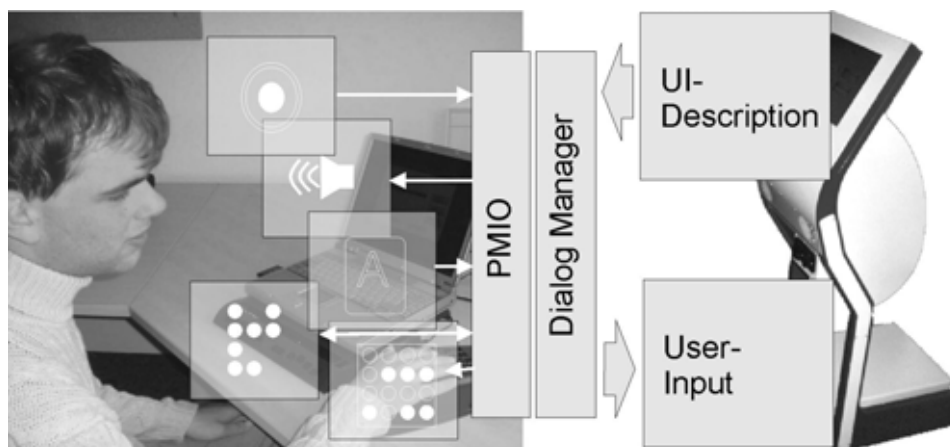


Abbildung 7.4: Multimodale Ein- und Ausgabe am Beispiel verschiedener Methoden für visuell eingeschränkte Nutzer (Sprache, Keyboard, Braille, taktile Ausgabe der Firma Handy-Tech).

Abbildung 7.4 zeigt ein Beispiel für die möglichen multimodalen Interaktionsmittel, welche durch das Darstellungssystem angesprochen werden können. Die verschiedenen modalitätsspezifischen Darstellungen und Eingaben werden in der Polymodalen Ein- und Ausgabe in Form des abstrakten User Interface Modells verwaltet und synchron gehalten. Die Darstellung wird von spezifischen Rendermodulen (entsprechend den Standard Design Prozessen) realisiert. Anpassung kann hierbei innerhalb der PMIO oder der Darstellungsmodule erfolgen. Dieses Vorgehen deckt sich mit der des Entwicklungsmodells. Eingaben werden in das User Interface modell eingetragen und in alle angeschlossenen Darstellungsmodule weitergegeben, so dass ein konsistenter Zustand aller Instanzen des Interfaces gewährleistet werden kann.

Kontextadaptivität Der modulare Aufbau der Darstellungspipeline in der Polymodalen Ein- und Ausgabe ermöglicht die Einbindung zusätzlicher Anpassungsprozesse. Konzeptuell wurden hier verschiedene Prozesse definiert die zum einen Transformationen, Analysen und Extraktionen von In-

formationen umfassen. Diese Prozesse werden in Form einer Pipeline zusammengeschlossen und können so frei konfigurierbar eine Nachbearbeitung und Analyse der Benutzerschnittstelle ermöglichen. Verschiedene Komponenten wurden im Rahmen des Projektes definiert welche auf die abstrakte Beschreibung zugreifen konnten um sich so kontextsensitiv zu konfigurieren. Eine virtuelle Tastatur nutzte die Information zur Anzahl der Elemente um verschiedene Scanmodi automatisch einzustellen. Eine Spracheingabe analysierte die Benutzerschnittstellen um dynamisch Grammatiken für die Eingabe zu generieren. Eine Sprachausgabe nutzte die Hilfs- und Fehlermeldungen um verschiedene Formen der Ausgabe zu generieren. Eine Wortergänzung ist in der Lage sich aufgrund des User Interface Kontext zu konfigurieren und die vorgeschlagenen Worte auf den inhaltlichen Kontext anzupassen.

Andere Formen der benutzerseitigen Konfiguration wurden ebenfalls berücksichtigt und umgesetzt. Für eine ausführliche Schilderung der Konzepte und Methoden soll an dieser Stelle auf [EMB03] bzw. [MR01, RH04b, Ric02a, Ric02b, Ric01, RE03, RH04a] verwiesen werden.

7.1.2 Benutzerstudie

Mehrere Studien wurden während der Laufzeit des Projektes durchgeführt um die Relevanz und die Benutzbarkeit der entwickelten Lösungen sicherzustellen. Das Darstellungssystem wurde insbesondere in der abschließenden Studie in seiner Umsetzung für verschiedene benutzerspezifische Assistenzgeräte validiert. Die hier beschriebene Studie wurde durch Mitarbeiter der Humboldt-Universität Berlin unter Leitung von Frau Marita Enge durchgeführt. Die hier vorgestellten Ergebnisse wurden zum Teil in [EM02, RE03, RH04a] veröffentlicht.

7.1.2.1 Mobile Assistenzgeräte

Das Darstellungssystem wurde als Teil des EMBASSI-Prototypen in verschiedenen Ausprägungen auf verschiedenen Assistenzgeräten eingesetzt.

Ein *mobiles Assistenzgerät für Nutzer ohne Einschränkungen* wurde auf einem Compaq iPAQ Pocket PC realisiert, welches über eine Java-basierte Anwendung gesteuert werden konnte. Die Darstellung erfolgte durch die Umsetzung der abstrakten Beschreibungssprache in HTML. Diese wurden dynamisch zur Laufzeit generiert.

Ein *mobiles Assistenzgerät für visuell eingeschränkte Nutzer* (s. Abbildung 7.1) wurde prototypisch auf einem Laptop realisiert. Zum derzeitigen Zeitpunkt (im Jahr 2002) waren keine miniaturisierten mobilen Endgeräte verfügbar, die die verwendete Assistenzsoftware (Spracherkennung, Screenreader) ausführen konnte. Der Anpassungsaufwand auf existierende Geräte hätte die verfügbaren Mittel überstiegen. Aufgrund der Entwicklung der Rechenleistung auf mobilen Geräten ist jedoch in absehbarer Zeit mit den entsprechenden Systemen zu rechnen, so dass die Übertragung der Ergebnisse auf wirklich mobile Endgeräte eingeschränkt zulässig erscheint. Die Konfiguration des Assistenzgerätes umfasste folgende Merkmale:

- Braille oder QWERTZ-Keyboard zur Eingabe
- Braille-Zeile zur Ausgabe und zur Steuerung des Fokus.
- Ein taktiles Display welches eine niedrig aufgelöste Darstellung graphischer Informationen mittels Piezo-elektronischer Elemente ermöglicht, z. B. zur Lokalisierung des Fokus auf dem Bildschirm oder zur Anzeige von Bildern.
- Drei verschiedene Sprachausgaben: *a)* Ausgabe der Beschriftung des aktuellen Elements; *b)* Ausgabe von Anmerkungen und Kurzhilfen; und *c)* Ausgabe von Fehlermeldungen.

Die Kombination von haptischem und akustischem Feedback erwies sich als geeignet, visuell eingeschränkten Nutzern umfangreiche Kontrolle über die Funktionen des Systems zu erhalten. Obgleich dieser frühe Prototyp noch kaum als mobil zu bezeichnen war, existieren in der Zwischenzeit Mobilgeräte wie der *Nokia Communicator* mit Braille-Zeile und Sprachausgabe, bzw. das *CX-1* von *Siemens* mit Sprachausgabe.



Abbildung 7.5: Eine gelähmte Person bei der Nutzung des Systems über eine Kinnsteuerung und ein virtuelles Keyboard, bzw. eine Bedieneinheit der Firma IBS Seveke.

Ein Assistenzgerät für körperlich eingeschränkte Benutzer 7.5 wurde ebenfalls aus den oben genannten Gründen auf einem Laptop installiert. Das Problem dieser Nutzergruppe liegt darin dass diese meist nur wenige Freiheitsgrade der Bewegung bewusst kontrollieren können und so auf minimale Eingabemethoden reduziert sind. Die Konfiguration des Assistenzgerätes wies daher folgende Merkmale auf:

- Eine Bedieneinheit, welche zwei oder drei Freiheitsgrade der Eingabe auf verschiedene Systemeingaben (Enter, Maus, etc.) abbildet. Die Einheit iteriert durch die verschiedenen Funktionen mit einer festen Geschwindigkeit. Die momentan ausgewählte Funktion wird durch ein Licht markiert und kann durch eine einfache Eingabe übernommen werden. Je nach Interface kann auch zwischen verschiedenen Funktionssätzen gewechselt werden.
- Spracheingabe für definierte Kommandos (welche aus dem Kontext des User Interfaces genommen werden) sind möglich. Diese können auch auf eine einfache Lautsteuerung reduziert sein.
- Eine virtuelle Tastatur ermöglicht in Kombination mit einer kontextsensitiven Wortergänzung die Eingabe von Texten über die Bedieneinheit [CHK⁺02].

Auf diese Weise können körperlich eingeschränkte Personen durch Spracheingabe und minimale physikalische Interaktionen mit graphischen Oberflächen und externen Geräten wie Terminals interagieren, wie dies ohne den mobilen Assistenten nicht der Fall ist.

Zusätzlich wurde ein Assistenzgerät für ältere Personen auf einem Acer Tablet PC installiert, welcher über die EMBASSI Plattform durch Stifteingabe und Sprachein- und ausgabe zu steuern war.

7.1.2.2 Methode

In einer Benutzerstudie wurden die Assistenzgeräte für körperlich eingeschränkte und für ältere Personen in dem oben geschilderten Einkaufsszenario getestet. Die Evaluation konzentrierte sich

auf die drei Hauptaspekte der DIN EN ISO 9241/11 [ISO01] Definition der Benutzbarkeit, welche unter anderem die Aspekte der Effektivität, Effizienz und Akzeptanz umfasst. Ziel der Studie war es, die folgenden drei Fragen zu beantworten:

1. *Frage 1:* Erlaubt das System körperlich eingeschränkten Benutzern mit Terminals mit der selben Einfachheit und Qualität zu interagieren wie nicht-eingeschränkte Nutzer?
2. *Frage 2:* Profitieren Nutzer von der personalisierten Assistenz?
3. *Frage 3:* Wie ist der Gesamteindruck und die Akzeptanz des Systems?

Das experimentelle Design bestand aus einem genesteten Design mit zwei Gruppen von Nutzern (körperlich Eingeschränkte, [physically handicapped = PH] vs. nicht eingeschränkte [non-disabled = ND]) um die erste Frage zu beantworten. Die ND Gruppe wurde in zwei Gruppen unterteilt, die entweder das Terminal direkt nutzen sollten (ND1) oder mit diesem über das Assistenzgerät für ältere Menschen interagieren sollten (ND2) um so die zweite Frage zu beantworten. Als abhängige Variable wurde die Effizienz über die Bearbeitungszeit und die Anzahl der Bearbeitungsschritte bestimmt. Zusätzlich wurde die Belastung mittels des NASA Task Load Index (NASA TLX) [HS88] erfasst. Akzeptanz wurde anhand eines neunskaligen Usability Fragebogens erhoben, der für diese Studie entwickelt wurde. Sechs Personen mit mit starken körperlichen Einschränkungen (2 weiblich / 4 männlich) und zwei Gruppen zu je zehn Personen ohne Behinderungen (je 4 weiblich / 6 männlich) nahmen an dieser Studie teil. Das durchschnittliche Alter betrug ca. 31 Jahre. Die Erfahrung mit Technik war in der Gruppe der nicht-behinderten Nutzer signifikant größer.

Teilnehmer der Experimentalbedingung (PH, ND2) wurden gebeten zwei Aufgabenzyklen durchzuführen. Jeder Zyklus bestand zunächst aus einem offline Teil in dem eine Einkaufsliste mittels des Einkaufslistenagenten ausgefüllt werden sollte (*Liste*), und einem online Teil bei dem diese Liste dann an den Terminal übermittelt werden sollte (*Kaufen*). Die Listen in beiden Zyklen hatten zwei Elemente gemein, so dass der Nutzen aus der Speicherung alter Ergebnisse (History Effekt) zum tragen kommen konnte.

7.1.2.3 Ergebnisse

Frage 1: Für die erste Aufgabe konnte in den Daten kein Unterschied in der Vollständigkeit (V) und Genauigkeit (G) bei der Aufgabenausführung zwischen Benutzern mit und ohne Mobilgerät festgestellt werden (alle Gruppen erreichten ca. 100% Genauigkeit und Vollständigkeit im ersten Durchgang). Jedoch konnte gezeigt werden, dass Nutzer ohne Assistenz im zweiten Durchgang deutlich weniger genau waren als im ersten (U-Test: $p < .002$), und ebenfalls weniger genau und vollständig als Nutzer mit mobiler Assistenz (U-Test: $p < .002$). Zwischen den beiden Gruppen mit Assistenz zeigte sich kein signifikanter Unterschied in beiden Durchgängen (s. Tabelle 7.1). In der online Bedingung konnte der Einkaufsassistent die Leistung der behinderten Nutzer deutlich verbessern (U-Test: $p = .046$) im Vergleich zu Nutzern ohne mobiler Assistenz.

(%)	Nicht Behinderte ohne Mobilgerät		Nicht Behinderte mit Mobilgerät		Behinderte mit Mobilgerät	
	V	G	V	G	V	G
Liste 1	100	100	100	100	100	100
Liste 2	73	65	98	90	100	92
Kaufen 1	99	91	100	94	100	98
Kaufen 2	100	91	98	94	100	100

Tabelle 7.1: Vollständigkeit (V) und Genauigkeit (G) der beiden offline (*Liste*) und online Durchgänge (*Kaufen*) für die drei Gruppen des Tests.

Frage 2: Um den Nutzen der Personalisierung durch die Einkaufsliste zu erfassen wurde die Ersparnis an Zeit und Interaktionsschritten zwischen dem ersten und zweiten Durchgang erhoben. Die

Analyse erfolgte auf Basis der Log-Dateien (s. Tabelle 7.2). Eine Zeitersparnis war für alle Gruppen zu beobachten, wenngleich diese nicht signifikant für die Gruppe der körperlich eingeschränkten Nutzer war. Weiterhin wurde ein Anstieg des Interaktionsaufwandes für die Assistenzbedingung beobachtet. Dieser kann aufgrund der komplexeren Interaktion mit dem User Interface der Assistenzkomponente entstanden sein. Der Assistenzeffekt bei den Zeiten war für die Gruppe der Behinderten zwar hoch, konnte jedoch keine signifikante Größe erreichen. Bei den Interaktionsschritten konnte ein deutlicher Assistenzeffekt beobachtet werden, dieser wurde jedoch nur für die körperlich eingeschränkten Nutzer signifikant (T-Test: $p = .04$). Mentale und motorische Anforderungen waren für alle Bedingungen gering (s. Abbildung 7.6). Nutzer des mobilen Assistenten (ND2) zeigten eine deutliche Verringerung physischer Anstrengung (T-Test: $p = .041$), während die Nutzung ohne mobile Assistenz geringere mentale Anstrengung bedeutete.

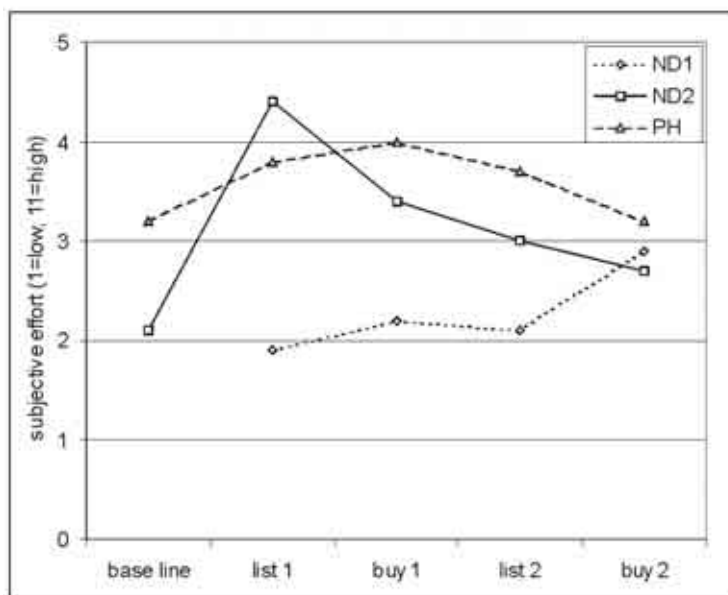


Abbildung 7.6: Subjektive Bewertung des Aufwandes.

	Nicht Behinderte ohne Mobilgerät	Nicht Behinderte mit Mobilgerät		Behinderte mit Mobilgerät	
Zeitersparnis (min)					
	LE	LE	AE	LE	AE
Liste		1.7	2.0	-1.1	3,1
Kaufen	0.6	1.4	1.1	-0.5	3,8
Reduzierung Arbeitsschritte					
	LE	LE	AE	LE	AE
Liste		5.5	2.9	-6.3	-3.8
Kaufen	2.8	12.6	7.9	-2.5	8,7

Tabelle 7.2: Zeitersparnis und Reduzierung des Aufwandes für die drei Bedingungen aufgeteilt in Lerneffekt (LE) und Assistenzeffekt (AE).

Frage 3: Die Akzeptanz wurde mittels eines neunskaligen Usability Fragebogens, der für diese Studie entwickelt worden war. Die durchschnittlichen Bewertungen, die in Abbildung 7.7 aufgeführt sind, können als durchgehend positiv gewertet werden.

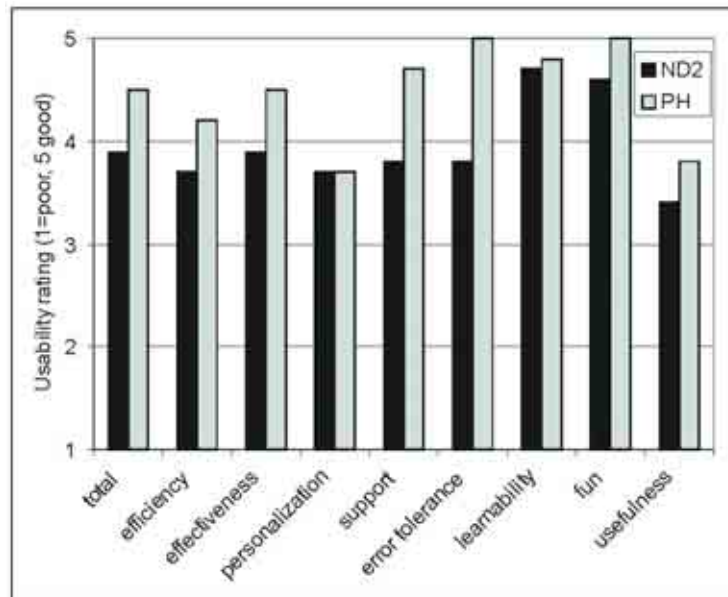


Abbildung 7.7: Bewertung der Benutzbarkeit

7.1.2.4 Diskussion

Die Ergebnisse der Studie konnten belegen, dass ein multimodales durch Assistenz unterstütztes System geeignet ist, Nutzern mit körperlichen Einschränkungen ein deutlich höheres Maß an Autonomie und Freiheit im Umgang mit öffentlichen Terminalsystemen zu ermöglichen.

- Selbst Personen mit starken körperlichen Einschränkungen konnten das Terminal ohne fremde Hilfe benutzen. Die Aufgabenausführung war fast genauso präzise und vollständig wie die nicht behinderter Nutzer.
- Obgleich die Konzepte, die in der Studie eingeführt wurden für die Benutzer ungewohnt waren konnte für alle Gruppen eine vergleichsweise geringe mentale Belastung festgestellt werden. Personen ohne Assistenz empfanden mehr Anstrengung als Personen mit.
- Physikalische Belastung konnte durch die Systeme verringert werden.
- Die Benutzbarkeit des Systems wurde als gut bis sehr gut bewertet. Dennoch zeigte sich, dass Benutzer ohne Behinderung keine Neigung zeigten das System auch selbst zu nutzen.
- Einen großen Nachteil stellten die langen Bearbeitungszeiten der Behinderten Benutzer dar. Allerdings zeichnete sich bereits nach wenigen Durchgängen ein klarer Assistenz bzw. Lerneffekt ab. Eine Überarbeitung der Benutzerschnittstelle des Assistentenmoduls könnte zu einer Reduzierung der Interaktionsaufwände führen.

Zunächst soll nochmals auf die Fragen, die zu Eingang dieser Studie aufgeworfen wurden eingegangen werden. Die Studie konnte folgende Antworten aufzeigen.

Antwort 1: Körperlich eingeschränkte Nutzer konnten mithilfe des Systems genauso gut oder sogar besser mit dem Terminalsystem interagieren als nicht behinderte Nutzer ohne Assistenz. Dies ist besonders erfreulich angesichts der Tatsache, dass diese Nutzergruppe ansonsten von der Bedienung von Terminals weitestgehend ausgeschlossen ist.

Antwort 2: Die Nutzung von Mobilgeräten führte nicht gleichzeitig zu einer Reduzierung des zeitlichen oder Interaktionsaufwands. Die Assistenz durch Personalisierung führte zu einer deutlichen

Erleichterung für nicht behinderte Nutzer. Die Assistenz des Einkaufslistenmoduls führte zu einer Zeitersparnis für alle Nutzer. Der deutliche Lerneffekt bei der Benutzung von Mobilgeräten weist jedoch darauf hin, dass das Potential dieser Anwendung insbesondere nach einiger Eingewöhnung zum tragen kommen kann.

Antwort 3: Benutzerakzeptanz war für die Mobilen Assistenten durchgängig positiv. Insbesondere die Unterstützung körperlich eingeschränkter Nutzer wurde als gute Unterstützung, fehlertolerant und unterhaltsam gewertet.

Die Studie offenbart die Umsetzbarkeit und den Nutzen des EMBASSI-Konzeptes der mobilen multi-modalen Assistenz. Durch das System konnten Menschen die ansonsten von der Nutzung von Terminalsystemen ausgeschlossen waren frei und präzise mit diesen interagieren. Das Darstellungssystem konnte in diesem Kontext zeigen, dass es die plattformübergreifende und multimodale Darstellung von Benutzerschnittstellen in verschiedenen Szenarien unterstützt und die Abbildung komplexer Interaktionen zulässt. Im Rahmen dieser Arbeit konnte somit die technische Machbarkeit, sowie die Benutzbarkeit der so erzeugten User Interfaces belegt werden.

7.2 Validierung der Transformationsstrategien

Die Transformationsstrategien, welche in den Abschnitten 5.2.3 und 5.3 eingeführt und in Abschnitt 6.2 in Form von Transformationsalgorithmen zur Umsetzung von User Interfaces zwischen Plattformen mit verschiedenen Displaygrößen umgesetzt wurden, werden in diesem Abschnitt in zwei Untersuchungen zur Wiedererkennung und zur allgemeinen Benutzbarkeit validiert.

1. *Überprüfung der Transformationshypothese:* die Annahme, dass Benutzer Regelmäßigkeiten des Transformationsprozesses nutzen können um das Aussehen der Variante zu antizipieren und Vorwissen auf diese Variante anwenden zu können ist die zentrale Annahme der Transformationshypothese. Diese musste zunächst überprüft werden um als Grundlage für weitere Untersuchungen zu dienen.
2. *Anwendung einer regelbasierten Transformation:* die praktische Umsetzbarkeit einer regelbasierten Transformation, und deren Vorteile in der Benutzbarkeit gegenüber manuell angepassten Interfaces wurde anhand einer kommerziell erhältlichen Anwendung belegt. Die regelbasiert angepasste Variante wurde mit der kommerziell erhältlichen Variante verglichen.

7.2.1 Überprüfung der Transformationshypothese

Diese erste Studie dient zunächst der Überprüfung der Transformationshypothese selbst (s. Abschnitt 5.2.3). Die Hypothese, welche hier überprüft werden soll besagt, dass der Benutzer bei einem Wechsel zwischen zwei Varianten einer Benutzeroberfläche, dies als eine Transformation der einen Variante in die andere wahrnimmt. Von besonderer Bedeutung sind hierbei zunächst die Aspekte der Darstellung. Die Hypothese besagt weiterhin, dass der Benutzer, stellt sich die Transformation als regelmäßig für ihn dar, diese Transformation auf sein mentales Modell des Dialoges anwendet.

Zur Überprüfung dieser Hypothese wurde der Effekt der *mentalen Rotation* verwendet. Dieser Effekt wurde von Cooper & Shepard [CS73] beschrieben und als Beleg für das Vorhandensein analoger Prozesse bei der visuellen Wahrnehmung und der menschlichen Informationsverarbeitung generell interpretiert [Pai86, EK93]. Die Anregung zu dieser Studie entsprang einem Vortrag von Jim Foley im *Haus der Graphischen Datenverarbeitung* in Darmstadt im Jahre 2004 [Fol04, BSF88]



Abbildung 7.8: Objekte aus dem Experiment zur mentalen Rotation von Cooper & Shepard, 1973.

Cooper & Shepard stellten den Probanden die Aufgabe zu entscheiden, ob sich zwei Objekte gleichen. Die Manipulation in diesem Test bestand darin dass die Objekte zueinander rotiert waren (s. Abbildung 7.8). In dieser Studie und vielen weiteren Studien die diesen Effekt replizierten, konnte festgestellt werden, dass die Antwortzeit der Probanden mit dem Winkel der Rotation zwischen den Objekten proportional anstieg. Die Reaktionszeit war bei 180° am längsten und fiel in beide Richtungen zu 0° und 360° hin linear ab. Dieser Effekt wurde dahingehend interpretiert, dass die Betrachter um ihr Vergleichsurteil zu bilden, die beiden Bilder ‚nebeneinander halten‘ müssen und hierzu eines der Objekte *mental rotierten*.

Kann dieser Effekt der mentalen Rotation nun bei der Rotation eines User Interfaces in Befolgung der Transformationsstrategien aus Abschnitt 5.3 nachgewiesen werden, dann ist damit ein Indiz für die Unterstützung der Transformationshypothese erbracht: die Transformation der Präsentation hat ihr Äquivalent in der mentalen Repräsentation des Benutzers.

Während bei Untersuchungen zu mentalen Rotation meist Vergleichsurteile abgefragt wurden, wurde in dieser Studie eine visuelle Suchaufgabe genutzt. Die Teilnehmer sollten eine Sequenz von Elementen zunächst in dem Original Interface anwählen und dann in einer rotierten Variante dies wiederholen.

Die Verwendung einer Suchaufgabe anstelle eines Vergleichsurteils hat zwei Gründe: Die Wahrnehmung eines kompletten Dialogs und die Beurteilung zweier Dialoge auf Ähnlichkeit ist im Vergleich zur üblichen Nutzung von Dialogen nicht sonderlich realistisch. In der Regel interagiert der Benutzer mit einem User Interface, daher sollte die Aufgabe Interaktion enthalten. Zum anderen zeigte Rock [Roc73] (zitiert nach [EK93]), dass der Effekt bei abstrakten und komplexen Objekten nicht so deutlich nachzuweisen ist. Eine Fokussierung auf einzelne Komponenten des Dialogs, wie dies bei einer Suche der Fall ist, hilft hier, die Komplexität zu reduzieren.

Als Zielreaktion wurde eine Sequenz von Eingaben verwendet. Grund hierfür ist die räumliche Ausrichtung der Sequenz. Eine Sequenz von Eingaben bildet ein räumliches Muster. Nutzt der Proband eine mentale Rotation zur Vorhersage der Lage der Zielelemente im rotierten Interface, kann er ebenso dieses Muster rotieren und somit die Richtung der Verbindungsstrecken antizipieren. Die Verwendung einer Sequenz von Eingaben sollte somit die Wahrnehmung des räumlichen Zusammenhangs stärken.

In den folgenden Abschnitten wird zunächst die methodische Umsetzung und dann die Ergebnisse dieser Studie dargestellt. Ein Diskussion der Ergebnisse findet sich im letzten Abschnitt dieser Studie.

7.2.1.1 Methode

Jeder einzelne Durchgang bestand aus zwei Teilen: zunächst wurde das Original-Interface (640 × 480 Pixel) gezeigt, dann die gedrehte Variante. Die Teilnehmer sollten in der Variante die Elemente in einer bestimmten Reihenfolge anwählen, die ihnen zuvor bei dem Original vorgegeben wurde. Der Grad der Drehung der Variante stellte hierbei die unabhängige Variable dar und bestand aus vier Rotationswinkeln: 0°, 90°, 180° und 270°. Die Varianten der User Interfaces waren händisch unter Zuhilfenahme eines Visual Designers erstellt worden. Die um 0° gedrehte Variante war zugleich das Original. In Abbildung 7.9 werden alle 5 Dialoge in den jeweils 4 Orientierungen dargestellt. Die Durchführung wurde in 4 Blöcke aufgeteilt. In jedem Block wurden vier der Dialoge jeweils viermal in einer Orientierung gezeigt. Als erstes wurde stets die aufrechte Version gezeigt, die darauf folgenden Blöcke wurden zufällig angeordnet ebenso wie die Sequenz der auszuwählenden Elemente. Tabelle 7.3 zeigt den zeitlichen Ablauf der Studie. Die zufällige Auswahl der Elemente sollte verhindern, dass der sonst üblicher Fluss der Interaktion bei der Rotation Einfluss nimmt. Aus demselben Grund wurden die Interfaces größtenteils ohne eine offensichtliche Hauptinteraktionsrichtung entworfen. Als Ausgangspunkt diente stets eine Schaltfläche am unteren Rand des umgebenden Testdialogs (in Abbildung 7.10 nicht sichtbar) um die Bewegungen einheitlich zu gestalten.

Die Aufgabe der Teilnehmer dieses Versuchs war es also zunächst, in dem, auf Knopfdruck erscheinenden Original-Dialog (0° gedreht) drei Elemente in der Reihenfolge anzuklicken, in der diese rot aufleuchteten. Ausgangslage war immer eine Schaltfläche am unteren Rand des Test-Dialogs um

Block	Original	Variante	Anzahl	Position
Block 1	0°	0°	4 × 4	Position 1
Block 2	0°	90°	4 × 4	Position 2,3, oder 4
Block 3	0°	180°	4 × 4	Position 2,3, oder 4
Block 4	0°	270°	4 × 4	Position 2,3, oder 4

Tabelle 7.3: Design der Studie. Die Anordnung der Dialoge innerhalb der Blöcke erfolgte zufällig, ebenso die Auswahl der Elemente und die Orientierung in Block 2 bis 4.

die Entfernung, die mit der Maus zurückgelegt wurde konstant zu halten. Wurde ein Element angeklickt, färbte es sich grün, wenn die Auswahl korrekt, und dunkelrot, wenn die Auswahl falsch war. Wurden drei Elemente angewählt, verschwand der erste Dialog und die Ausgangsschaltfläche erschien wieder, diesmal grün. Mit dieser Schaltfläche wurde nun der zweite Teil gestartet, hier erschien die Variante des Dialogs, ohne dass die Elemente aufleuchteten. Der Teilnehmer sollte die Elemente aus der vorherigen Ansicht so schnell wie möglich in derselben Reihenfolge anwählen. Er erhielt erneut Rückmeldung über Fehler, nach drei Klicks verschwand der Dialog und der Durchgang war beendet.

Die Zeiten für die einzelnen Eingaben wurden automatisch erfasst, ebenso wie die Fehler.

Neben der unabhängigen Variablen Orientierung wurden folgende Maße kontrolliert um die Ergebnisse auf Artefakte zu kontrollieren [Car03b]:

- *Anzahl der Elemente (n)*: Da es sich bei der Aufgabe um eine Suchaufgabe handelte musste das Hick-Hyman-Gesetz [Hic52, Hym53] beachtet werden, welches einen logarithmischen Zusammenhang zwischen der Reaktionszeit (RT) und der Anzahl der zu suchenden Elemente, bzw. Antwortalternativen vorhersagt.

$$RT = a + b \log_2(n)$$

Aus diesem Grund wurde die Zahl der Elemente in den Dialogen konstant gehalten.

- *Schwierigkeit (ID)*: Nach dem Gesetz von Fitts [Fit54, FP64] kann die Reaktionszeit bei der Auswahl eines Elements in einem Dialog durch seine Schwierigkeit, oder *item difficulty (ID)* vorhergesagt werden. Diese berechnet sich in Abhängigkeit von der Entfernung (A) die zurückgelegt werden muss und der Größe des Elements (W). In dieser Studie wurde die Gleichung nach Shannon [Mac89] verwendet, diese lautet:

$$ID = \log_2(A/W + 1)$$

Da dieser Faktor nicht konstant gehalten werden konnte, wurde er erfasst und die Reaktionszeit mittels regressionsanalytischer Verfahren bereinigt.

Der Versuchsaufbau bestand aus einer Java-Anwendung, die auf einem Apple iBook installiert war. Die Auflösung des TFT Bildschirms war 1024 × 768 Pixel. Die Teilnehmer benutzten eine handelsübliche optische Maus um mit der Anwendung zu interagieren. Die Aufgabe für die Teilnehmer bestand aus einer Suchaufgabe. Abbildung 7.10 zeigt die Versuchsanwendung.

7.2.1.2 Ergebnisse

Sieben Personen (2 weiblich und 5 männlich) nahmen an der Studie teil. Insgesamt 480 Durchgänge wurden erfasst (zu Beginn führten zwei Teilnehmer Blöcke zu 20 Durchgängen durch).

Wie Abbildung 7.11 zeigt, dass der erwartete Unterschied in der Reaktionszeit für die verschiedenen Orientierungen in erster Linie bei der Zeit für die ersten Eingabe, bzw. bei der Pause zwischen den Durchgängen zu beobachten war. Um Störungen und Verzögerungen in den Zeiten durch die Teilnehmer aus den Daten zu eliminieren wurden zunächst 25 Durchgänge ausgeschlossen, bei denen es bei der ersten Eingabe zu Fehlern gekommen war. Weiterhin wurde auf die verbleibenden Werte ein Ausschlusskriterium von 90% angesetzt, so dass 52 Werte, die darüber lagen ausgeschlossen wurden. 428 Durchgänge blieben in der Folge für die Analyse erhalten.

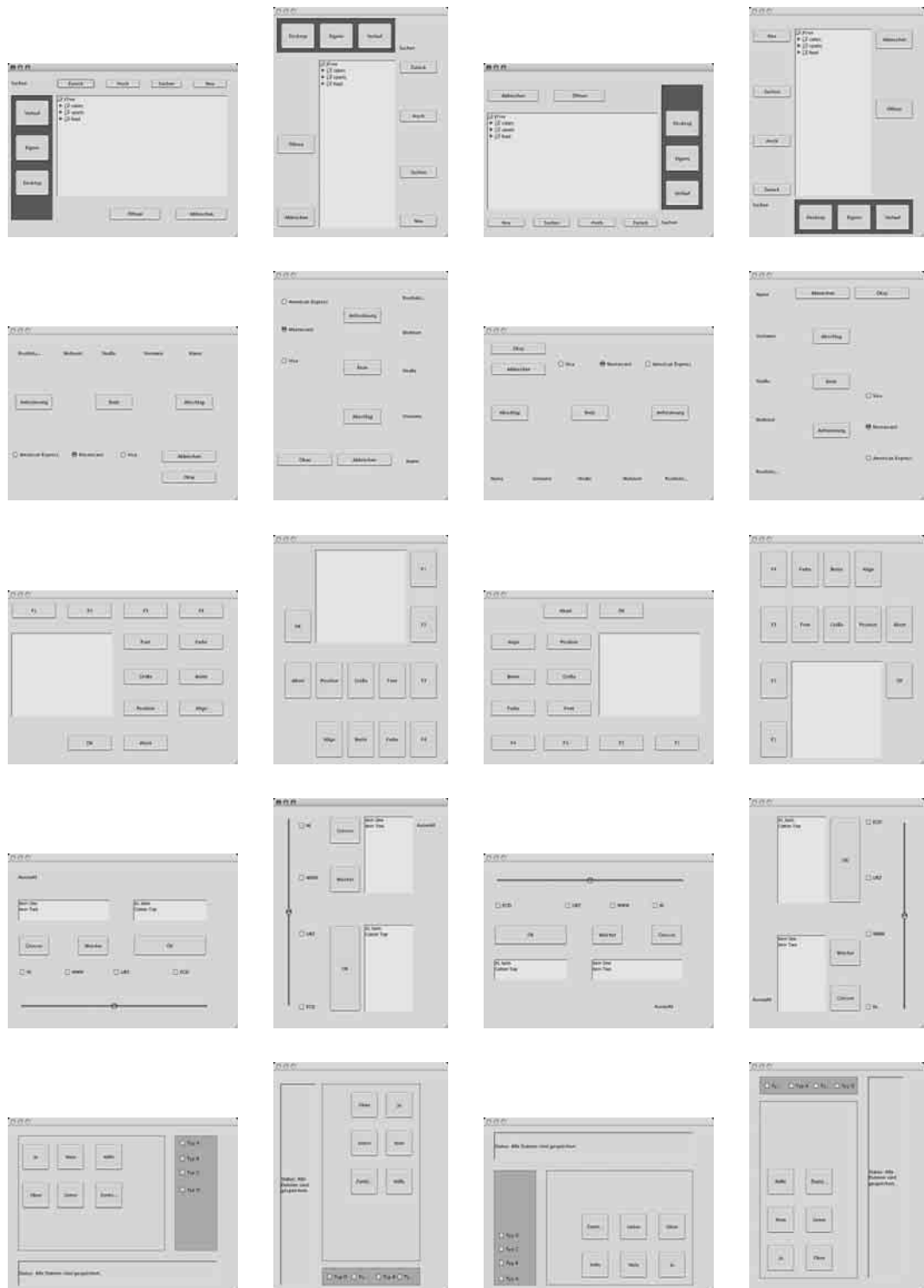


Abbildung 7.9: Abbildung der Dialoge in den verschiedenen Orientierungen (0°, 90°, 180°, 270°).

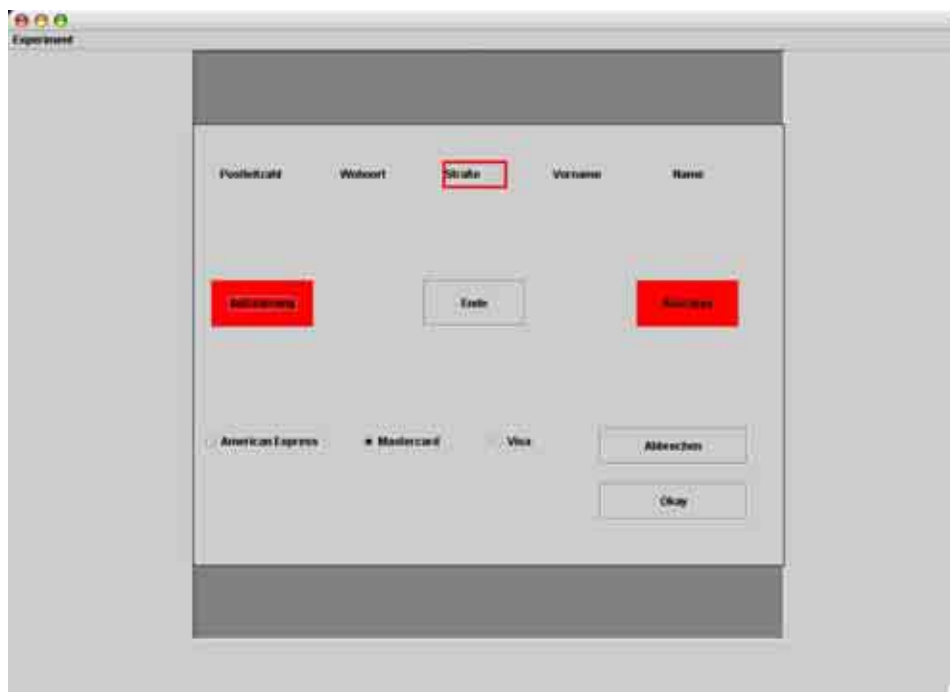


Abbildung 7.10: Ansicht der Versuchsanwendung mit den rot markierten Elementen des ersten Teils eines Durchgangs.

Analyse der Reaktionszeiten für die erste Eingabe: Die Analyse der Reaktionszeiten für die erste Eingabe wurde zunächst mittels einer schrittweisen linearen Regression durchgeführt. Ziel dieser Analyse war es die wichtigsten Faktoren, durch welche die Reaktionszeit beeinflusst wurde zu isolieren. Tabelle 7.4 zeigt das Ergebnis dieser Analyse. Der wichtigste Faktor war dieser Analyse zufolge die Position des Durchgangs im Block (*TRIAL*, 1 bis 16). Hierbei handelt es sich vermutlich um einen Lerneffekt. Dieser Lerneffekt überlagerte nach ca. sieben Durchgängen den Effekt der Mentalen Rotation. Der zweite Faktor ist der Schwierigkeitsindex (*ID*) nach Fitts und als dritter Faktor trägt die Variation der Orientierung, hier insbesondere der Kontrast der 90° und 270° rotierten Varianten gegenüber der 180° rotierten Variante signifikant zur Aufklärung der Varianz bei. Dieser Effekt unterstützt die Transformationshypothese.

Parameter	α -Niveau	R^2
Position (<i>TRIAL</i>)	.0015	.0325
Fitts Schwierigkeit (<i>ID</i>)	.0012	.0652
Orientierung (270° & 90°-180°)	.0181	.0822

Tabelle 7.4: Ergebnis der schrittweisen linearen Regressionsanalyse

Die Regressionsanalyse lieferte die Parameter, die zur Schätzung der Reaktionszeiten anhand der oben genannten Faktoren Schwierigkeit, Position und Orientierung notwendig waren. Eine lineare Gleichung zur Schätzung der Reaktionszeiten wurde aufgestellt:

$$RT_{est} = 1698.15 - 22.61 \times TRIAL + 1070.63 \times ID$$

Die Residuale dieser Schätzung, d.h. die Differenz der geschätzten Werte zu den realen Werten, ist nun um die Varianzanteile der Position und bereinigt. Diese Residuale wurden nun auf den Effekt der Orientierung hin untersucht. Abbildung 7.12 zeigt die Verteilung der Residuale über die Orientierungen. Die Verteilung zeigt die durch die Transformationshypothese vorhergesagten Form.

Der paarweise Vergleich (Student's t-Test) der Werte zeigt einen zum Teil signifikanten Effekt der Rotation auf die Reaktionszeit. Dieser ist freilich zwischen 0° und allen anderen Varianten hoch

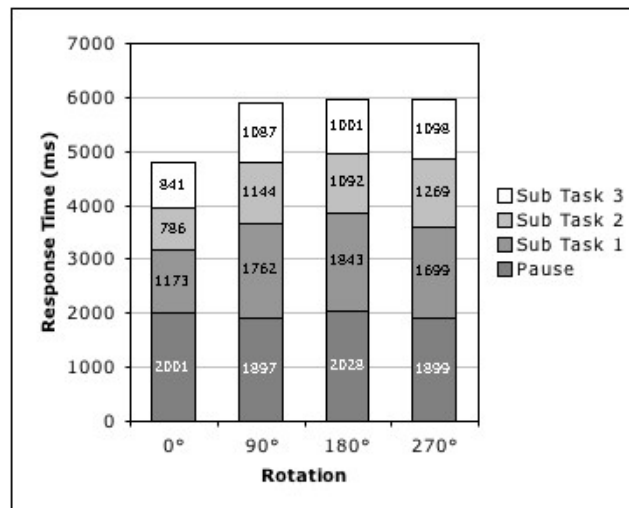


Abbildung 7.11: Die Verteilung der Reaktionszeiten über die einzelnen Eingaben und die vier Orientierungen. Die Abbildung zeigt, dass ein Effekt der Orientierung in erster Linie bei der ersten Eingabe und bei der Pause zu beobachten ist.

signifikant ($p < .0001$). Auch der Unterschied zwischen 90° und 180° ($t[DF = 415] = -1.47$; $p = .072$) und 180° und 270° ($t[DF = 415] = -2.63$; $p = .004$) ist deutlich bis signifikant, nicht jedoch zwischen 90° und 270° ($t[DF = 415] = 1.186$; $p = .11$).

Analyse der Pausenzeiten: Abbildung 7.11 zeigt erstaunlicherweise nicht nur während der Ausführung der ersten Teilaufgabe, sondern auch in der Dauer der Pausen einen Effekt der Orientierungen. Um diesen Effekt näher zu untersuchen wurde zunächst erneut eine Regressionsanalyse zur Aufklärung der relevanten Faktoren durchgeführt. Auch hier wurde ein klarer Effekt der Position (TRIAL) gefunden, wobei die Teilnehmer einen deutlichen Lerneffekt zeigten. 9 Extremwerte (>95%) wurden ausgeschlossen. Die folgende Regressionsgleichung wurde zur Vorhersage der Pausenzeiten verwendet ($F[1;469] = 25.26$; $p < .0001$):

$$RT_{est} = 2213 - 31 \times TRIAL$$

Abbildung 7.13 zeigt die Verteilung der Residuale der Pausenzeiten für die drei rotierten Bedingungen. Hier ist der Effekt zwar zu beobachten, kann jedoch nicht klar von einem zufälligen Effekt abgegrenzt werden (90° vs. 180°: $t[DF = 467] = -1.83$; $p = .03$; 180° vs. 270°: $t[DF = 467] = -1.28$; $p = .10$).

Weitere Ergebnisse Abbildung 7.14 zeigt die bereinigten Reaktionszeiten für alle drei Elemente der Eingabesequenz. In diesem Fall wurde die lineare Regression nur für die rotierten Versionen der Dialoge durchgeführt. Aus diesem Grund weichen die absoluten Werte in diesem Fall von den Werten der Regression über alle Orientierungen (s. Abb. 7.12) leicht ab. Die Verteilung der Reaktionszeiten zeigt eine Verschiebung des Musters weg von dem Effekt der Mentalen Rotation, bei dem der 180° gedrehte Dialog die längste Reaktionszeit zeigt, hin zu dem umgekehrten Muster. Während bei der zweiten Eingabe eine fast kontinuierliche Steigung hin zu 270° zu beobachten ist, zeigt die 180° Variante bei der dritten Eingabe die kürzeste Reaktionszeit.

Weiterhin scheint der Effekt der Mentalen Rotation nicht für alle Dialoge gleich stark ausgeprägt zu sein. Dies lässt auf eine Wechselwirkung mit der Gestaltung des Design der einzelnen Dialoge schließen. Eine genauere Untersuchung dieses Effektes konnte im Rahmen dieser Studie nicht durchgeführt werden und bleibt somit eine Aufgabe zukünftiger Studien.

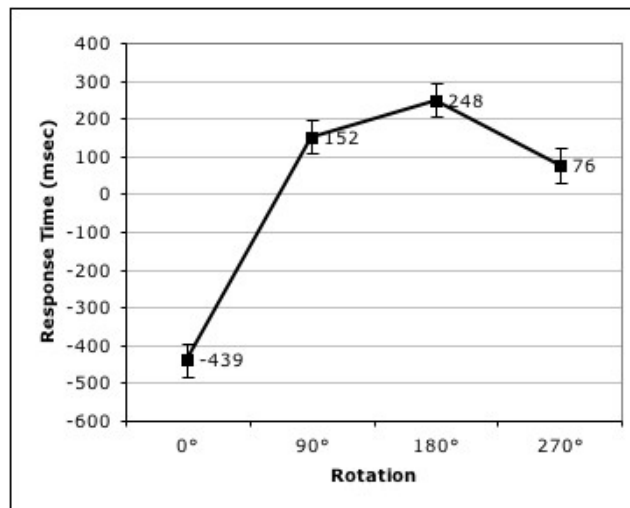


Abbildung 7.12: Verteilung der bereinigten Reaktionszeiten über die Orientierungen.

7.2.1.3 Diskussion

Die Ergebnisse der Studie unterstützen die Transformationshypothese. Auch wenn der Rotations-effekt erwartungsgemäß nicht sehr stark ausfällt, so ist dennoch eine längere Reaktionszeit für 180°rotierte Varianten im Vergleich zu den 90° und 270°rotierten Varianten zu beobachten. Die Teilnehmer des Versuchs scheinen die Rotation, die innerhalb der einzelnen Blöcke konsistent auf die verschiedenen User Interfaces angewendet wurde, genutzt zu haben, um die Lage des Zielelements mittels einer mentalen Transformation zu antizipieren.

Der Effekt äußert sich nicht nur in der aktiven Zeit (Eingabe 1) sondern auch in der Pausenzeit, wenngleich auch nicht signifikant. Die längeren Pausenzeiten in der vorhergesagten Richtung könnten als Hinweis dafür gewertet werden, dass ein Teil der mentalen Rotation bereits vorbereitend stattfindet.

Die kontrollierten Maße, insbesondere das Schwierigkeitsmaß nach Fitts, hatten einen deutlichen Anteil an den beobachteten Reaktionszeiten. Mithilfe statistischer Verfahren gelang es, diese Effekte zu isolieren und anhand der bereinigten Werte einen Effekt der Rotation nachzuweisen. Diese Lösung wurde gewählt, da die Alternative, eine Konstanzhaltung der Schwierigkeit über die Orientierungen kaum praktisch umsetzbar gewesen wäre, ohne den Suchraum künstlich einzuschränken. Insbesondere der Effekt der Versuchssequenz (TRIAL) weist auf einen deutlichen Lerneffekt hin. Der Rotationseffekt wird kontinuierlich geringer über die Anzahl der Wiederholungen hinweg, bis er nach ca. sieben Durchgängen verschwindet. Es scheint ein Wechsel in der Strategie des Benutzers aufzutreten. Die Strategie des Nutzers scheint von der Transformation der originalen Repräsentation hin zur Verwendung einer neuen, separaten Repräsentation der modifizierten Darstellung zu wechseln. Zum einen kann dies bedeuten, dass der Rotationseffekt nur ein Effekt ist, der in neuen Situationen auftritt und somit bei trainierten Anwendungen keine Rolle mehr spielt und daher zu vernachlässigen ist. Zum anderen kann aber auch argumentiert werden, dass nur die konsistente Transformation einen derartigen Lerneffekt ermöglicht hat. Eine genauere Untersuchung dieses Effektes ist geplant.

Die große Differenz zwischen den unrotierten und den rotierten Varianten lässt darauf schließen, dass noch weitere Prozesse ablaufen, deren Ursprung noch erklärt werden muss. Ein weiteres Indiz hierfür ist die Tatsache, dass bei den Reaktionszeiten für die zweite und dritte Eingabe die Zeiten zwar erwartungsgemäß stark abfallen, jedoch eine deutliche Überlegenheit der 270°Rotation gegenüber 90° und 180° zu beobachten ist. Tendenziell scheint bei diesen Eingaben die Dauer für die 180°Drehung schneller zu gehen als bei den Portrait-orientierten Dialogen. Auch die Unterschiede, welche zwischen den Dialogen einer Orientierung auftraten, lassen auf weitere Effekte schließen, welche in folgenden Untersuchungen betrachtet werden sollten. Der Effekt von Gestaltungskriterien

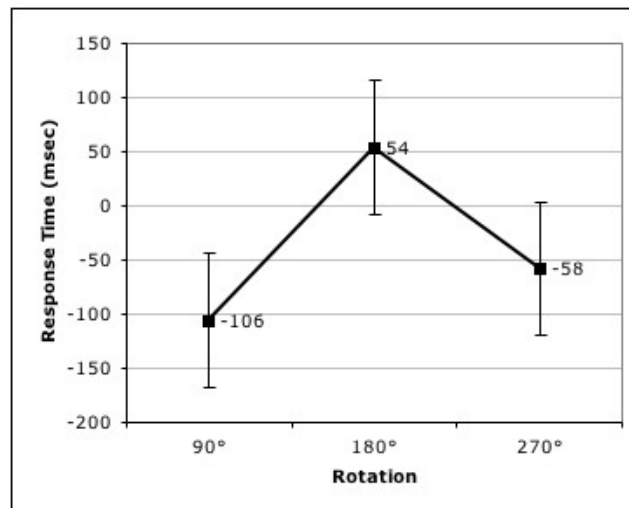


Abbildung 7.13: Verteilung der bereinigten Pausenzeiten für die drei gedrehten Dialoge.

wie Gruppierungen, Akzentuierung, Komplexität oder Balance [VG94] sollte ebenfalls eingehender untersucht werden.

In Abschnitt 7.2.1.1 wurde diskutiert, weswegen eine Sequenz von Eingaben anstatt einer einzelnen Eingabe gewählt wurde. Die Annahme war, dass eine Sequenz, aufgrund der räumlichen Eigenschaft der Bewegungspfade mit der Maus ebenfalls von der Antizipation durch eine mentale Rotation profitieren kann. Die Motivation, ein räumliches mentales Muster aufzubauen kann, so der Gedanke, unterstützt werden. Die Wahrnehmungsprinzipien der Gestalt-Psychologen (s. Abschnitt 4.4.1.4) können hierfür eine Erklärung geben. Nach dem Gesetz der *uniform connectedness* [PR94] werden ähnliche Elemente zu einer Form verknüpft. Die Hervorhebung der Zielelemente mit roter Farbe kann eine derartige perzeptive Verknüpfung provoziert haben und somit zum Aufbau eines mentalen Musters geführt haben. Der Vergleich zwischen den Zeiten für die einzelnen Eingaben (s. Abb. 7.14) scheint diese Annahme zu stützen. Der Effekt der mentalen Rotation kehrt sich über die zweite und dritte Eingabe um. Die horizontale Abbildung (von vielen Probanden auch als Spiegelung wahrgenommen) scheint hier nun schneller zu bearbeiten als die beiden anderen Darstellungen.

Unterschiede zwischen den verschiedenen Dialoge in der Charakteristik der Reaktionszeitverteilung konnten beobachtet werden. Bei manchen Dialogen waren die Rotationseffekte deutlicher als bei anderen. In einer Folgeuntersuchung werden die ausschlaggebenden Eigenschaften der einzelnen Dialoge detaillierter untersucht und experimentell variiert werden.

Ziel dieses Experimentes war es, zu untersuchen, ob ein Effekt der mentalen Rotation bei der Transformation eines User Interfaces nach den oben beschriebenen Transformationsstrategien zu beobachten ist. Die Ergebnisse dieser Studie stützen diese These. Dies eröffnet eine neue Perspektive auf das Problem der Konsistenz bei der Gestaltung von User Interfaces. Konsistenz kann somit nicht mehr nur als Beziehung zwischen zwei Instanzen eines Dialogs und dessen Unterelementen definiert werden, sondern ebenso als Konsistenz der Transformation zwischen den beiden. Zwei unterschiedliche Varianten eines User Interfaces können somit dennoch konsistent in Bezug auf die Transformation zwischen beiden sein. In Hinblick auf das Problem der Konsistenzmessung plattformübergreifender Anwendungen bietet diese Erkenntnis neue Möglichkeiten und stützt die Methode, welche in Abschnitt 5.4 beschrieben, in Abschnitt 5.4 umgesetzt, und in Abschnitt 7.3 schließlich überprüft wird.

7.2.2 Anwendung einer regelbasierten Transformation

In den meisten Fällen werden Entwickler eine Kombination der oben beschriebenen Transformationsstrategien anwenden, wenn sie eine Benutzerschnittstelle manuell an ein neues Gerät anpassen.

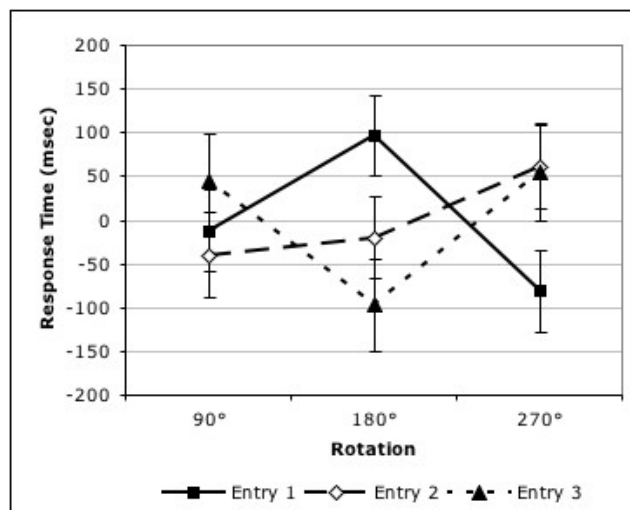


Abbildung 7.14: Verteilung der bereinigten Reaktionszeiten über alle drei Eingaben.

Automatische Layout Mechanismen [BCCR04, NML04, GVRV02, LVC02, VFO01, GW04] allerdings gewinnen schnell an Komplexität wenn sie mehrere Strategien gleichzeitig evaluieren und anwenden müssen. In diesem Abschnitt wird eine kombinierte Anwendung mehrere Transformationsstrategien vorgestellt und evaluiert, welche in erster Linie *Skalierung*, Segmentierung und Substitution einsetzt um die Symbolleiste und Menüleiste einer komplexen Office Anwendung—Microsoft Word—auf die Verwendung auf dem Pocket PC anzupassen [Ric05a].

Menüs und Symbolleisten enthalten einen erheblichen Anteil der Funktionalitäten einer Anwendung. In einigen Anwendungen (z. B. Textverarbeitung, Graphikprogramme) muss der Benutzer aus diesem Grund sehr häufig auf diese Elemente zugreifen. Nun ist es allerdings eine gängige Praxis (zumindest der Entwickler bei Microsoft) bei der Anpassung auf ein kleineres Gerät insbesondere bei der Symbolleiste Platz einzusparen. Die Unterstützung des Benutzers bei der Aufrechterhaltung eines konsistenten mentalen Modells, so nehmen wir an, sollte dabei helfen zwischen den Plattformen zu wechseln und die Einarbeitungszeit zu verringern. Dieser Effekt sollte bei häufig trainierten Interaktionen im Bereich des regel- und analogiebasierten Handelns mit vertrauten Anwendungen deutlich zu Tage treten [And95, HNM94]. Die Bereitstellung von automatisierbaren Mechanismen zur Anpassung, welche dies beachten sollte zum einen helfen, Entwicklungszeit zu reduzieren und zum anderen die Konsistenz der Transformation verschiedener Anwendungen erhöhen (z. B. die angepassten Versionen von Microsoft Excel, Word und Powerpoint könnten so anwendungsübergreifend konsistent gestaltet werden und so ein anbieterspezifisches Identifikationskriterium schaffen).

7.2.2.1 Darstellung der Anpassungsstrategie

Im folgenden Abschnitt wird eine Transformationsstrategie orgestellt, welche die automatische Transformation und Reduzierung von Menüs und Symbolleisten beim Wechsel zwischen Desktop PCs und Pocket PCs ermöglicht¹. Beide Geräte unterscheiden sich erheblich in ihrem verfügbaren Display-Platz, Interaktionsstil und Standards bezüglich Anordnung und Darstellung von Interfaceelementen und Menüs. Tabelle 7.5 gibt einen kurzen Überblick über die Unterschiede zwischen den Standardeigenschaften beider Geräte.

Neben den offensichtlichen Unterschieden in Display-Größe und Eingabemöglichkeiten existieren weitere Unterschiede in der Oberflächenkonzeption, welche ein Entwickler zu beachten hat. Während ein Dialog auf dem Desktop PC mehrere Symbolleisten enthalten kann, lässt der Pocket PC nur eine Symbolleiste zu. Auf dem Pocket PC sind Menüs am unteren Rand des Bildschirms angeordnet,

¹Microsoft Windows, Pocket PC, Microsoft Word, Microsoft PocketWord und Microsoft Office sind Registrierte Markenzeichen der Microsoft Inc.

Desktop PC	Pocket PC
ca. $1280 \times 1024px$	$240 \times 320px$
Keyboard, Maus	Stift
Menü oben	Menü unten
0 – n Symbolleisten	0 – 1 Symbolleisten

Tabelle 7.5: Standard Eigenschaften der beiden Plattformen.

während sie auf dem PC am oberen Rand des Fensters liegen. Auf dem Pocket PC teilen sich Symbolleiste und Menü denselben — knappen — Bereich am unteren Rand des Bildschirms, während sie auf dem Desktop PC mehrere Zeilen umfassen können und prinzipiell getrennt dargestellt werden. Dies schränkt den Umfang der Symbolleiste auf dem Pocket PC dadurch natürlich im Vergleich zur Desktop Plattform deutlich ein.

Die Anpassungsstrategie beim Wechsel von der Desktop Plattform auf die kleinere Pocket PC Oberfläche wird in drei grundlegende Schritte eingeteilt:

- **Menü:** So lange nur ein Menü und keine Toolbars vorhanden sind, muss der Transformationsalgorithmus lediglich den verfügbaren Platz auf dem Bildschirm mit der Anzahl der Menüelemente auf oberster Ebene verglichen. Entweder können dann alle Menüelemente gleichzeitig angezeigt werden, oder für den Fall, dass zu viele Elemente im Menü sind, muss dieses angepasst werden (Ein Pocket PC bietet Platz für ca. sechs Elemente). Eine mögliche Anpassungsstrategie ist hierbei von rechts her Elemente in ein Untermenü (z.B. mit dem Namen „Weitere“) zu verlegen bis die verbleibenden Elemente angezeigt werden. In der Regel ordnen sich Menüelemente in Übereinstimmung mit der Leserichtung von links nach rechts in absteigender Wichtigkeit.
- **Toolbar:** Analog zu dem obigen Beispiel—wenn also nur eine Toolbar in dem Dialog vorhanden ist—überprüft der Algorithmus ob alle Toolbar-Elemente angezeigt werden können (hier sind auf dem Pocket PC ca. neun Elemente nebeneinander möglich). Ist dies nicht der Fall gibt es zwei möglich Anpassungsstrategien:
 - a) Gibt es nur wenige überzählige Schaltflächen in der Toolbar, wird eine scrollbare Toolbar verwendet (s. Abb. 7.15a).
 - b) Gibt es mehr als doppelt so viele Elemente als darstellbar sind, oder müssen mehrere Toolbars dargestellt werden, muss eine umschaltbare Toolbar eingesetzt werden (s. Abb. 7.15b). Der Algorithmus versucht, alle Elemente einer Toolbar in eine Gruppe zu packen, ist dies nicht möglich, werden sie auf mehrere Gruppen verteilt.

In beiden Fällen können Toolbar-Elemente über die Schaltflächen zur Steuerung nach links oder rechts erreicht werden, für die scrollbare Variante werden dabei die Elemente einzeln weiter geschaltet, in der umschaltbaren Version wird immer um komplette Gruppen weiter geschaltet. Oberhalb der Toolbar liegt ein schmaler Streifen mit einem Balken, der die aktuelle Position angibt. Über diesen Streifen kann über Tippen ebenfalls gescrollt oder umgeschaltet werden. Eine alternative Version der umschaltbaren Toolbar ermöglicht das Umschalten über beschriftete Schaltflächen (s. Abb. 7.17).

- **Menü und Toolbar:** In den meisten Anwendungen stehen sowohl Menü als auch Toolbar zur Verfügung. Häufig stellen beide dieselben Funktionen zur Verfügung, wobei diese über die Toolbar meist einfacher und schneller zugänglich sind. Die Anpassung eines komplexen Interfaces für ein kleines Endgerät kann daher häufig beide Strategien benötigen. Abbildung 7.16 zeigt einen Ausschnitt aus einem komplexen Interface einer Standard Office Anwendung (Microsoft Word) und 7.17 zeigt, wie dieses Interface automatisch auf ein kleineres Endgerät angepasst werden könnte indem die modifizierte Version der umschaltbaren Toolbar eingesetzt wird.

Algorithmus 2 Transformations Algorithmus

```
1: procedure TRANSFORMATION
2:    $d \leftarrow Display.width$ 
3:    $M \leftarrow \{m_1, m_2, \dots, m_n\}$ 
4:    $T \leftarrow \{t_1, t_2, \dots, t_m\}$ 
5:   if  $T = \emptyset$  then
6:     PUTMENU
7:   else if  $M = \emptyset$  then
8:     PUTTOOLBAR
9:   else
10:    PUTMENU
11:    PUTTOOLBAR
12:  end if
13: end procedure
14: function PUTMENU
15:   if  $M \cup T \leq d$  then
16:     put menu
17:   else
18:     collapse
19:   end if
20: end function
21: function PUTTOOLBAR
22:   if  $T \cup M \leq d$  then
23:     put toolbar
24:   else if  $T \leq 2 \times d$  then
25:     put scrollable toolbar
26:   else
27:     put switchable toolbar
28:   end if
29: end function
```

7.2.2.2 Anwendung von Transformations Strategien auf eine Office Anwendung

Zur Validierung der in Abschnitt 5.3 beschriebenen Transformationsstrategien wurden diese zu Anpassung einer gängigen Office Anwendung, nämlich Microsoft Word, auf ein Pocket PC Gerät eingesetzt. Da eine Pocket PC-Version dieser Anwendung fester Bestandteil der Pocket PC Systems ist, konnte das anhand der Transformationsalgorithmus (jedoch händisch) abgeleitete Interface mit der existierenden von Experten manuell angepassten Version verglichen werden. Abbildung 7.17 zeigt das Ergebnis der Transformation.

Ausgehend von der Microsoft Word Anwendung für den Desktop PC und den in der Pocket PC verfügbaren Funktionen (die reduziert sind) wurde die Transformation in den folgenden drei Schritten durchgeführt:

1. *Reduktion*: Nicht alle Funktionen, die auf dem Desktop PC verfügbar sind stehen auch auf dem Pocket PC bereit. Anhand der Funktionen, die in der existierenden Anwendung realisiert sind wurde der Funktionssatz reduziert.
2. *Umschaltbar Toolbar*: Da die Anzahl der Toolbars und auch der Toolbar-Elemente auch nach der Reduktion die darstellbare Zahl klar überschritten wurde eine umschaltbare Toolbar verwendet. Es wurde dabei darauf geachtet, dass die Gruppen, die in der Desktop Version der Anwendung zusammengefasst sind auch in der Pocket PC-Version zusammen blieben. Weiterhin wurde den Gruppen Namen gegeben, die dann auf die Schaltflächen zum Umschalten geschrieben wurden (s. Abb. 7.17). Die Namen wurden von den Funktionen der Gruppen abgeleitet.
3. *Menü*: Da auch das Menü deutlich den verfügbaren Platz überstieg wurden alle Menüelemente unter den Menüpunkt „Menü“ zusammengefasst.

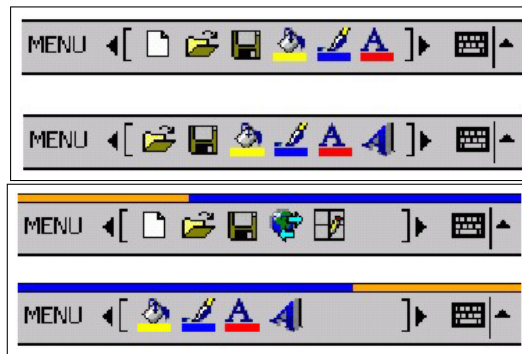


Abbildung 7.15: Die scrollbare (*links*) und die umschaltbare (*rechts*) Toolbar.

Abbildung 7.18 zeigt das ursprüngliche Interface (C) der Microsoft PocketWord Anwendung und Abbildung 7.17 zeigt das Ergebnis der der regelbasierten Transformation (A). Einige der wesentlichen Unterschiede zwischen beiden Versionen sind:

- In A ist die Toolbar, den Standards der Plattform folgend in die Menüleiste integriert und ist stets sichtbar und muss nicht gesondert geöffnet werden, wie die in C der Fall ist.
- In C existiert nur eine Toolbar mit acht Elementen während in A vier Sätze zu je sieben Schaltflächen und Comboboxen identisch mit den Elementen der Desktop Version enthalten sind.
- In C ist das Menü im Vergleich zur Desktop Version umgeordnet und erheblich gekürzt worden. Die Grundstruktur des Menüs ist hier deutlich verändert worden und Funktionen wurden in anderen Teilen des Menüs platziert während in A sowohl Toolbar als auch Menü dieselbe Struktur und Reihenfolge wie die Desktop Version haben.
- In C existieren Menü-Elemente auf oberster Ebene, die direkt Funktionen auslösen (z.B. ‚Neu‘), dies ist nicht der Fall in A.

7.2.2.3 Benutzerstudie

Um die Ergebnisse der regelbasierten Transformation mit der normalen Version zu vergleichen und die erwarteten Effekte in der Benutzbarkeit belegen zu können wurde eine empirische Studie durchgeführt. Es wurde erwartet, dass die regelbasiert angepasste Version zum einen den Plattform-Standards stärker entspricht und zum anderen eine konsistentere Abbildung der Desktop-Anwendung darstellt als die normale PocketWord Anwendung. Daher sollte es dem Benutzer einfacher möglich sein sein mentales Modell der Anwendung zu übertragen und somit weniger Probleme bei der Übersetzung zu haben. Aus diesem Grund sollte die transformierte Version zumindest genauso attraktiv und benutzbar sein wie die, von menschlichen Experten angepasste PocketWord Anwendung. Aufgrund der konsistenten Abbildung sollte die transformierte Version als der Desktop-Version ähnlicher empfunden werden.



Abbildung 7.16: Ein Ausschnitt der Microsoft Word Anwendung.



Abbildung 7.17: Die umschaltbare Toolbar auf einem Pocket PC.

7.2.2.4 Methode

Die Teilnehmer ($n = 12$, 3 female/9 male) wurden gebeten, sieben Aufgaben auf beiden Versionen, der klassischen (C) und der alternativen (A) zu erfüllen. Folgende Aufgaben wurden gestellt:

1. Erzeugen sie ein neues Dokument.
2. Formatieren sie den Text fett.
3. Setzen sie den Font auf Courier New.
4. Kopieren sie und fügen sie einen Satz ein.
5. Formatieren sie den Text rechtsbündig.
6. Sichern sie das Dokument als...
7. Nutzen sie die „Wörter zählen“ Funktion.

Beide Anwendungen wurden zum Teil in .Net neu programmiert um so Zugang zu allen Funktionen und Ereignissen zu erhalten und so die Benutzereingaben mitloggen zu können. Zeiten und Eingaben wurden in Logdateien geschrieben. Nach jedem Test wurde die übersetzte Version des Software Usability Scale (SUS) [Bro96] von den Teilnehmern ausgefüllt um so eine subjektive Einschätzung der Benutzbarkeit zu erhalten. Um Reihenfolgeeffekte zu vermeiden, wurden die Teilnehmer zufällig einer der beiden Reihenfolgegruppen ($A \rightarrow C$ und $C \rightarrow A$) zugeordnet ($n(C) = 6$; $n(A) = 6$). Erfahrung im Umgang, Geschlecht und Alter wurden kontrolliert, wobei insbesondere Erfahrung gleichmäßig über die Bedingungen verteilt war. Alle Personen hatten viel Erfahrung im Umgang mit Word auf dem Desktop PC.



Abbildung 7.18: Die Microsoft PocketWord Anwendung.

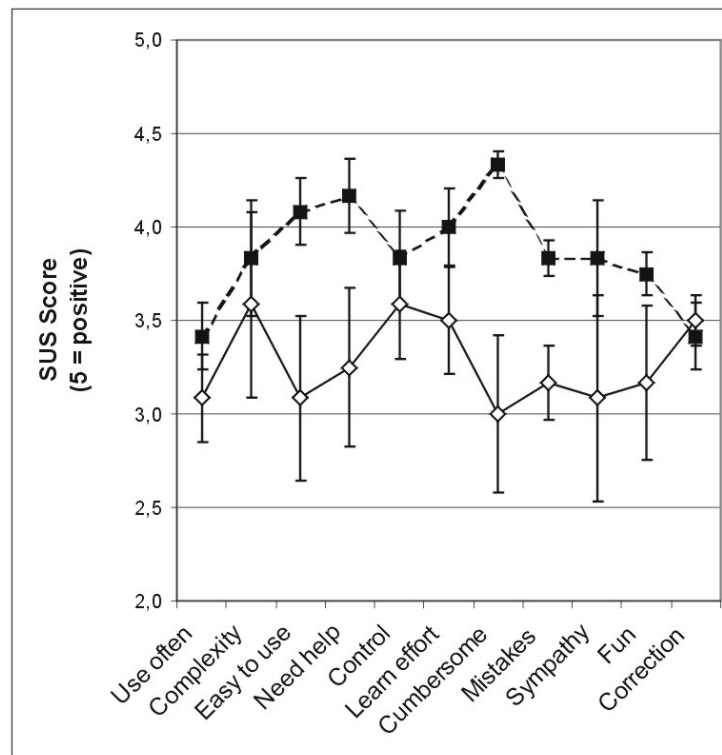


Abbildung 7.19: Die Akzeptanzwerte des SUS, hohe Werte stehen für gute Bewertungen (Alternative Version = schwarze gefüllte Quadrate; Klassische Version = weiße Rauten).

7.2.2.5 Ergebnisse

Eingabezeiten und SUS-Bewertungen wurden statistisch analysiert. Im folgenden werden die Ergebnisse einzeln vorgestellt.

Akzeptanz Abbildung 7.19 zeugt die Ergebnisse der Akzeptanzbewertungen beider Anwendungen. Ein einseitiger t-Test für abhängige Stichproben zeigt, dass die Bewertung für die Alternative Version in fast allen Kriterien besser bewertet wurde als die klassische. Die Ergebnisse zeigen, dass die Teilnehmer der Meinung waren, die alternative Version sei leichter zu benutzen ($p=.03$) und lernen ($p=.034$), leichter zu interagieren ($p=.003$), und weniger fehleranfällig ($p=.002$) sowie etwas unterhaltsamer ($p=.055$ n.s.) als die klassische Version.

Ähnlichkeit Alle Teilnehmer sollten im Anschluss an beide Durchgänge beurteilen, welche der Versionen der Desktop Anwendung ähnlicher sei. Zehn von zwölf Teilnehmern beurteilten die alternative Version als deutlich ähnlicher (ein Teilnehmer war sich nicht sicher und einer war der gegenteiligen Meinung). Die meisten unerfahrenen Teilnehmer waren überrascht, dass die alternative Version nicht die eigentliche, von Microsoft vertriebene Anwendung war. Als Grund für die größere Ähnlichkeit wurden die ähnlichen Icons und die ähnliche Anordnung der Toolbar-Elemente genannt. Ebenfalls wurde die konsistente Anordnung der Menüelemente als ähnlicher wahrgenommen.

Bearbeitungszeiten Abbildung 7.20 zeigt die Differenzen der Bearbeitungszeiten für beide Geräte. Da die Zeiten sehr stark streuten mussten extreme Werte die größer als zwei Standardabweichungen über dem individuellen Durchschnitt lagen ausgeschlossen werden. Die Analyse der Werte zeigt, dass in den meisten Fällen die alternative Version schneller bearbeitet werden konnte als die klassische. Insbesondere häufige Eingaben wie Formatierung und Speichern zeigten eine bessere Bearbeitungszeit. Dennoch gab es auch einige Aufgaben, bei denen die alternative Version deutlich

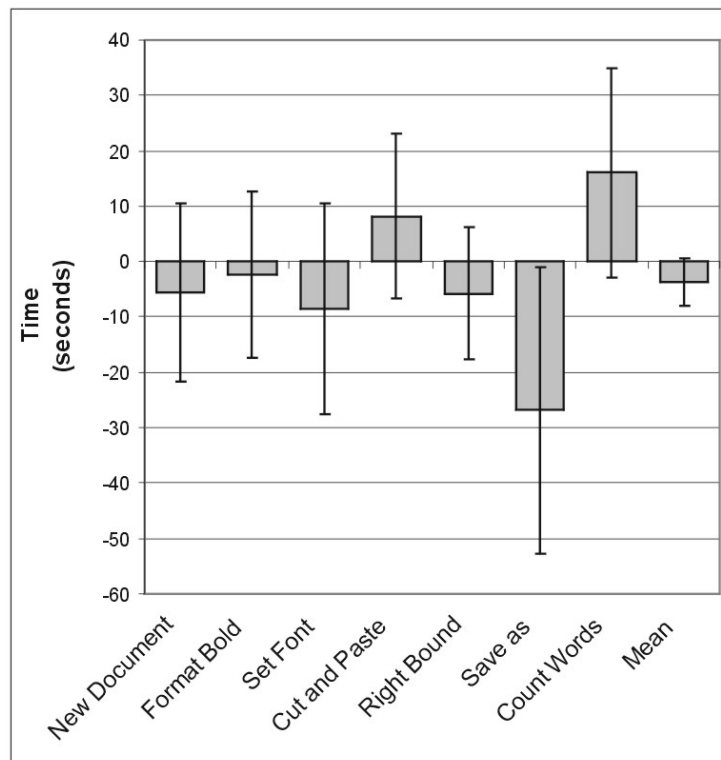


Abbildung 7.20: Die Differenzen in den Bearbeitungszeiten pro Aufgabe in Sekunden ($t_A - t_C$; negative Werte stehen für eine schnellere Bearbeitung in der Alternativen Version).

Aufgabe	Dauer (ms)	t-Test (1-seitig)	p
Neues Dokument	-5574	1.193	0.1
Format Fett	-2348	0.538	0.3
Font setzen	-8608	-1.616	0.067
Ausschneiden	8186	-1.822	<i>0.049</i>
Rechtsbündig	-5718	1.669	0.0617
Speichern	-26838	3.598	<i>0.002</i>
Wörter zählen	16040	-2.937	<i>0.007</i>

Tabelle 7.6: Bearbeitungszeiten, fettgedruckte Ergebnisse sind signifikant.

schlechter war (s. Tabelle 7.6). Im Mittel war die Bearbeitung auf der alternativen Version jedoch um 6 Sekunden signifikant schneller ($p = .007$).

7.2.2.6 Diskussion

In Übereinstimmung mit unseren Erwartungen erwies sich die alternative Version mit der regelbasiert transformierten Darstellung und der umschaltbaren Toolbar als benutzbarer, schneller und der Desktopanwendung ähnlicher als die kommerziell verfügbare Version von PocketWord.

Dieses Ergebnis stützt die Annahmen, welche in dieser Arbeit beschrieben wurden dahingehend, dass eine konsistente Transformation den Wechsel bei der Benutzung derselben Anwendung auf verschiedenen Endgeräten vereinfacht. Zusätzlich scheint es so, dass nicht nur die Performanz, sondern auch die subjektive Einschätzung der Benutzbarkeit durch ein konsistentes Design positiv beeinflusst wird. Für die Herstellung von Anwendungen, die auf verschiedenen Endgeräten eingesetzt werden können kann diese Erkenntnis genutzt werden um so ein plattformübergreifendes *Corporate Design* zu entwickeln und Kunden durch die verbesserte plattformübergreifende Benutzbarkeit an die eigenen Produkte zu binden.

7.3 Validierung des Konsistenzmaßes

In diesem Abschnitt wird die experimentelle Validierung des in Abschnitt 5.4 entwickelten Konzeptes zur Messung der plattformübergreifenden Konsistenz beschrieben.

Das Konsistenzmaß leitet sich, wie in Abschnitt 5.4 beschrieben, aus der Transformationshypothese ab, und bietet die Möglichkeit, zum einen die Konsistenz innerhalb einer Anwendung auf einer Plattform, zum anderen aber auch die Konsistenz derselben Anwendung auf verschiedenen Plattformen zu berechnen. Basierend auf diesem Maß können Methoden zur Unterstützung plattformübergreifender Designs entwickelt werden, welche unmittelbar in den Entwicklungsprozess eingreifen und den Designer leiten [Ric06].

Das Konsistenzmaß baut auf der Annahme auf, dass die Regelmäßigkeit einer Abbildung einer Benutzerschnittstelle zwischen verschiedenen Endgeräten den lateralen Transfer von Wissen positiv beeinflusst. Konsistenz wird hierbei definiert als die Vorhersagbarkeit der Ausprägung eines Merkmals auf der zweiten Plattform aufgrund dessen Ausprägung auf der ersten Plattform (s. Abschnitt 5.4).

Singley und Anderson [SA85, SA89] untersuchten den lateralen Transfer von Anwendungswissen zwischen verschiedenen Formen von Texteditoren (s. Abschnitt 3.1). Sie ließen Personen nach einem fest vorgegebenen Plan über den Verlauf von mehreren Tagen Texte mit einem bestimmten Texteditor verfassen. Dann sollten diese Personen auf einen zweiten Editor wechseln. Sie konnten nachweisen dass die erworbene Expertise in einem Texteditor Auswirkungen auf die Benutzung des zweiten Editors hat. Wissen, so Singley und Anderson konnte umso besser transferiert werden, je mehr Produktionen wiederverwendet werden konnten.

In Anlehnung an das oben beschriebene Experiment wurde die Validierung des Konsistenzmaßes durchgeführt wobei der Wechsel zwischen Plattformen anstatt des Wechsels zwischen Anwendungen im Vordergrund stand. Aufgabe war es, eine einfache formbasierte Anwendung zunächst auf einem Desktop PC und anschließend auf einem Pocket PC zu verwenden. Die Konsistenz der Anwendung auf dem Pocket PC wurde variiert. Es wurde erwartet dass der Transfer zwischen den Geräten, und damit die Performanz auf dem Pocket PC in direkter Abhängigkeit zur Konsistenz steht.

Der Versuchsaufbau umfasste eine natürliche Interaktion mit einem Software System auf zwei Plattformen. Dieser Aufbau wurde gewählt um möglichst unverfälschten Einblick in die Effekte eines Wechsels zwischen Plattformen zu erhalten, ohne die Interaktion durch experimentelle Kontrollen übermäßig einzuschränken. Verschiedene Störeinflüsse, die auch die individuellen Unterschiede in der motorischen Umsetzung der Stifteingabe, perzeptive Unterschiede in der Erfassung der kleinen Interaktionsfläche und kognitive Unterschiede in der Umsetzung der Aufgabe betreffen, können

daher die zusätzliche Varianz in die Ergebnisdaten bringen. Zu erwarten war daher, dass die statistische Güte der somit erhaltenen Messwerte für statistisch valide Aussagen unter Umständen nicht ausreichen würden, vielmehr ein ökologisch valider Eindruck und Trend erfasst werden konnte. Die subjektive Evaluierung der verschiedenen Setups würde diesen Eindruck abrunden.

7.3.1 Methode

Die Versuchspersonen ($n = 15$) wurden aus den Mitarbeitern des Instituts rekrutiert. Es konnte somit von einer homogen recht hohen Vorerfahrung mit Computern ausgegangen werden. Dies ist zum einen sinnvoll um die Varianz gering zu halten, zum anderen kann davon ausgegangen werden, dass mobile Endgeräte bislang meist von Personen mit einer gewissen Vorerfahrung, häufig professionellen Anwendern genutzt werden.

Der Versuch wurde als einfaktorielles Design mit vier Gruppen angelegt. Die Gruppen unterschieden sich in der Konsistenz der Abbildung der von ihnen verwendeten Anwendung auf dem Pocket PC. Tabelle 7.7 zeigt die Verteilung der Versuchspersonen auf die verschiedenen Gruppen.

Konsistente Abbildung	Mittel inkonsistente Abbildung ($C = .6$)	Hoch inkonsistente Abbildung ($C = .2$)	Konsistente Transformation (Spiegelung)
-----------------------	--	--	--

Tabelle 7.7: Testbedingungen

Als Testanwendung wurde eine einfache formbasierte fiktive Anwendung zur Erfassung von Fahrzeugen für einen Fuhrpark eingesetzt. Diese Anwendung wurde in Microsoft .Net erstellt und lief mit identischen Anwendungskern mit einer Benutzerschnittstelle für den PC und vier verschiedenen Darstellungen für den Pocket PC. Auf allen Plattformen bestand die Anwendung aus einer Übersichtsmaske, eine Bearbeitungsmaske und vier Masken zur Bearbeitung der Angaben zu Verantwortlichem, Parkplatz, Marke und Fahrzeugtyp. Die Daten waren so gestaltet, dass keine langen Texte einzugeben waren um die Bearbeitung auf dem Pocket PC nicht unnötig zu erschweren. Die Anwendung war so gestaltet, dass die Benutzbarkeit mittelmäßig eine typischen Formularanwendung entsprach. Es war häufig nötig in Unterdialoge zu wechseln um Aspekte wie Fahrzeugtypen neu anzulegen.

Die Pocket PC User Interfaces unterschieden sich in der Konsistenz der Abbildung gemessen mit dem transformationalen Konsistenzmaß.

- Die *konsistente Abbildung (REG)* ($C = 1.0$) war eine direkte Umsetzung des Desktop PC User Interfaces. Die Positionierung aller Elemente erfolgte absolut konsistent. Die komplexe Haupt-Maske wurde auf drei Tabulatoren verteilt. Abbildung 7.22 zeigt eine Übersicht der Masken.
- Die *mittel inkonsistente Abbildung (MED)* ($C = 0.6$) war eine leicht modifizierte Umsetzung, bei der einige Elemente (ca. jedes zweite) in eine andere Position (eine von zwei möglichen) verschoben wurde (Abbildung 7.23). Eine exakte Halbierung der Konsistenz war aufgrund der eingeschränkten Anzahl der Masken nicht möglich, diese sollte jedoch zugunsten der geringeren Komplexität nicht erhöht werden.
- Die *hoch inkonsistente Abbildung (HIG)* ($C = 0.2$) war eine stark modifizierte Umsetzung, bei der jedes Element in der Position variierte, dabei aber keinem festen Muster gehorchte. Zusätzlich war die Reihenfolge der Tabulatoren vertauscht (Abbildung 7.24).
- Die *konsistent transformierte gespiegelte Abbildung (MIR)* ($C = 1.0$) war wiederum eine direkte Umsetzung des original Interfaces. Es stellte eine um eine horizontale Achse gespiegelte Abbildung der konsistenten Variante dar (Abbildung 7.25).

Zunächst sollte eine jede Person fünf Fahrzeuge aus eine Liste von Fahrzeugen, die auf Papier gedruckt vorlagen, eingeben. Abbildung 7.21 zeigt einen Ausschnitt aus diesen Testdaten. Die Darstellung war soweit abstrahiert, dass keine direkte Übertragung, sondern eine mentale Vorverarbeitung




MB1 (#113)	 	
		
Verantwortlicher	Mey (Abt. B2)	
Parkplatz	P1 (West, sicher)	
Typ	Kombi	
Marke	Mercedes-Benz	
Klimaautomatik	Ja	
Radio	Ja	
Standheizung	Nein	
CD	Ja	

Abbildung 7.21: Beispiel für die Fahrzeugdaten.

der Angaben notwendig war. Sowohl die Präsentation als auch die Positionierung der Daten unterschied sich deutlich von der Form, wie sie in die Anwendung eingegeben werden mussten. Auf diese Weise sollte eine intensivere Beschäftigung sowohl mit der Aufgabe als auch mit der Anwendung erreicht werden. In dieser Phase sollte vor allem erreicht werden, dass sich der Benutzer mit der Anwendung vertraut macht. Rückfragen und Hilfestellungen wurden aus diesem Grund ausführlich gegeben. Jeder Benutzer konnte so nach kurzer Zeit schnell und fehlerfrei arbeiten.

Nach fünf Fahrzeugen auf dem Desktop PC wurde die Person gebeten einen Usabilityfragebogen, den SUS [Bro96], auszufüllen, um neben den automatisch erfassten Bearbeitungszeiten auch ein subjektives Maß für die Benutzbarkeit der Anwendung zu erhalten. Im Anschluss daran wurde dem Versuchsteilnehmer ein Compaq iPAQ Pocket PC übergeben auf dem eine der vier Varianten der Anwendung lief. Nach einer kurzen Einführung sollten dann weitere fünf Fahrzeuge eingepflegt werden. Anschließend wurde erneut ein SUS ausgefüllt. Alle Teilnehmer bearbeiteten dieselben Fahrzeuge in derselben Reihenfolge.

7.3.2 Ergebnisse

Sollte die Konsistenz, wie sie durch das transformationale Konsistenzmaß erfasst wird einen Einfluss auf den Transfer von Anwendungswissen haben, dann sollten die Benutzer der konsistenten Anwendungen von diesem Vorwissen profitieren können und die Aufgabe entsprechend schneller bearbeiten können.

Aufgrund des Designs der Studie sind nun drei Abstufungen der Hypothese zu überprüfen:

1. *Qualitative Hypothese:* Die Bearbeitungszeit für die konsistente Anwendung ($REG < MED$; $REG < HIG$) ist kürzer als für die inkonsistenten Anwendungen.
2. *Quantitative Hypothese:* Ein linearer Zusammenhang zwischen dem transformationalen Konsistenzwert und der Bearbeitungszeit kann nachgewiesen werden ($REG < MED < HIG$).
3. *Transformationale Hypothese:* Ist lediglich die Positionsveränderung ausschlaggebend, dann ist die Bearbeitungszeit für die gespiegelte Variante genauso hoch, wie für die inkonsisten-

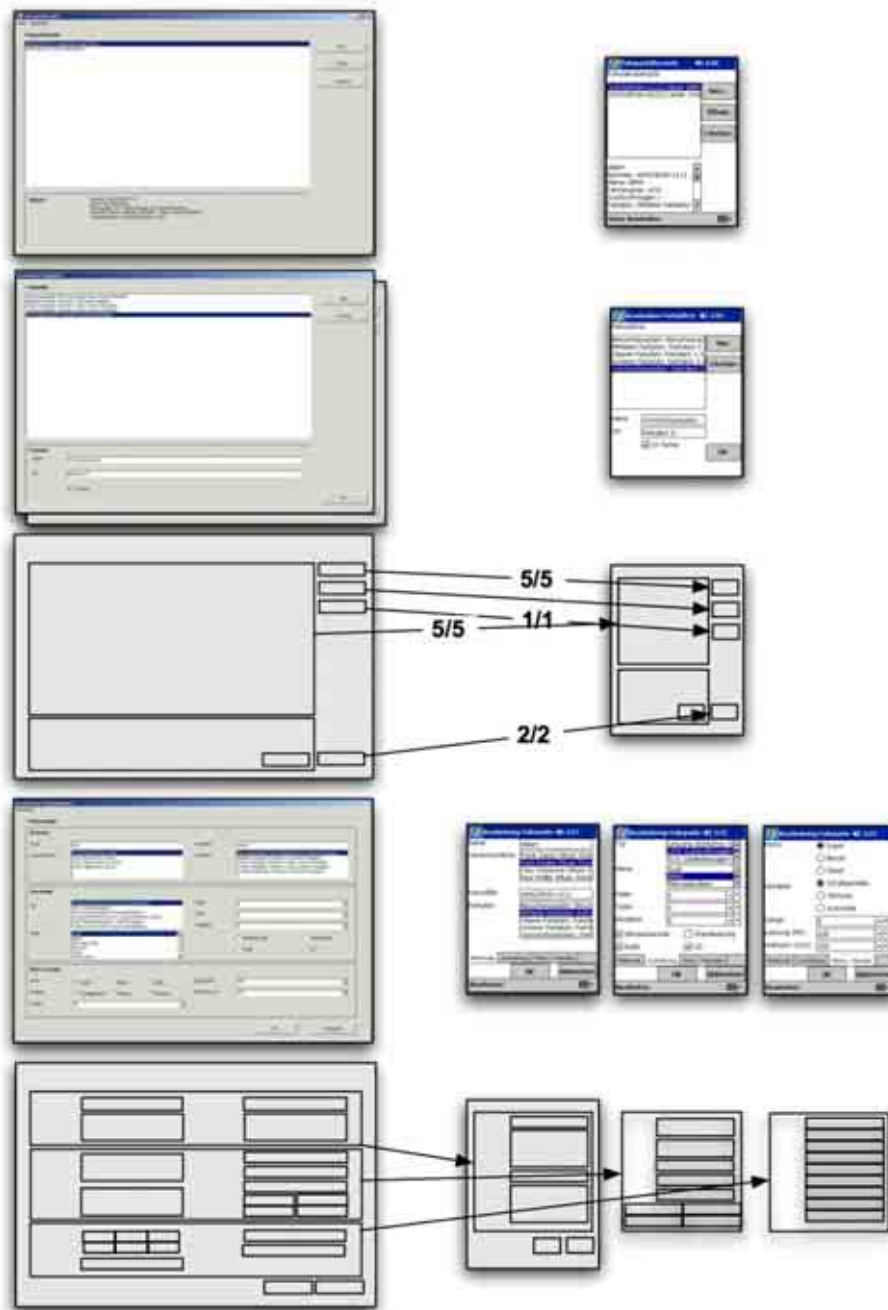


Abbildung 7.22: Beispiel für die konsistente Abbildung

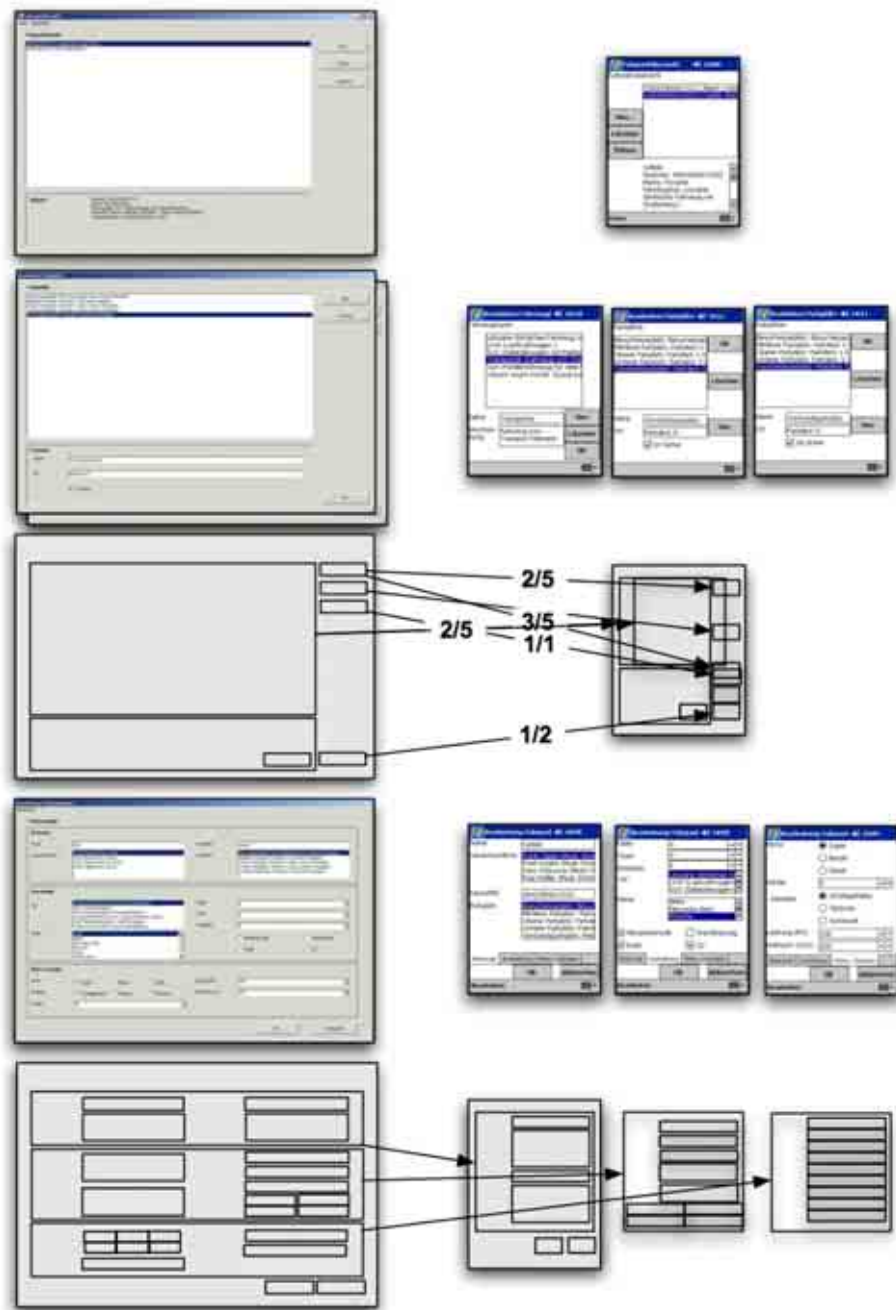


Abbildung 7.23: Mittel konsistente Abbildung

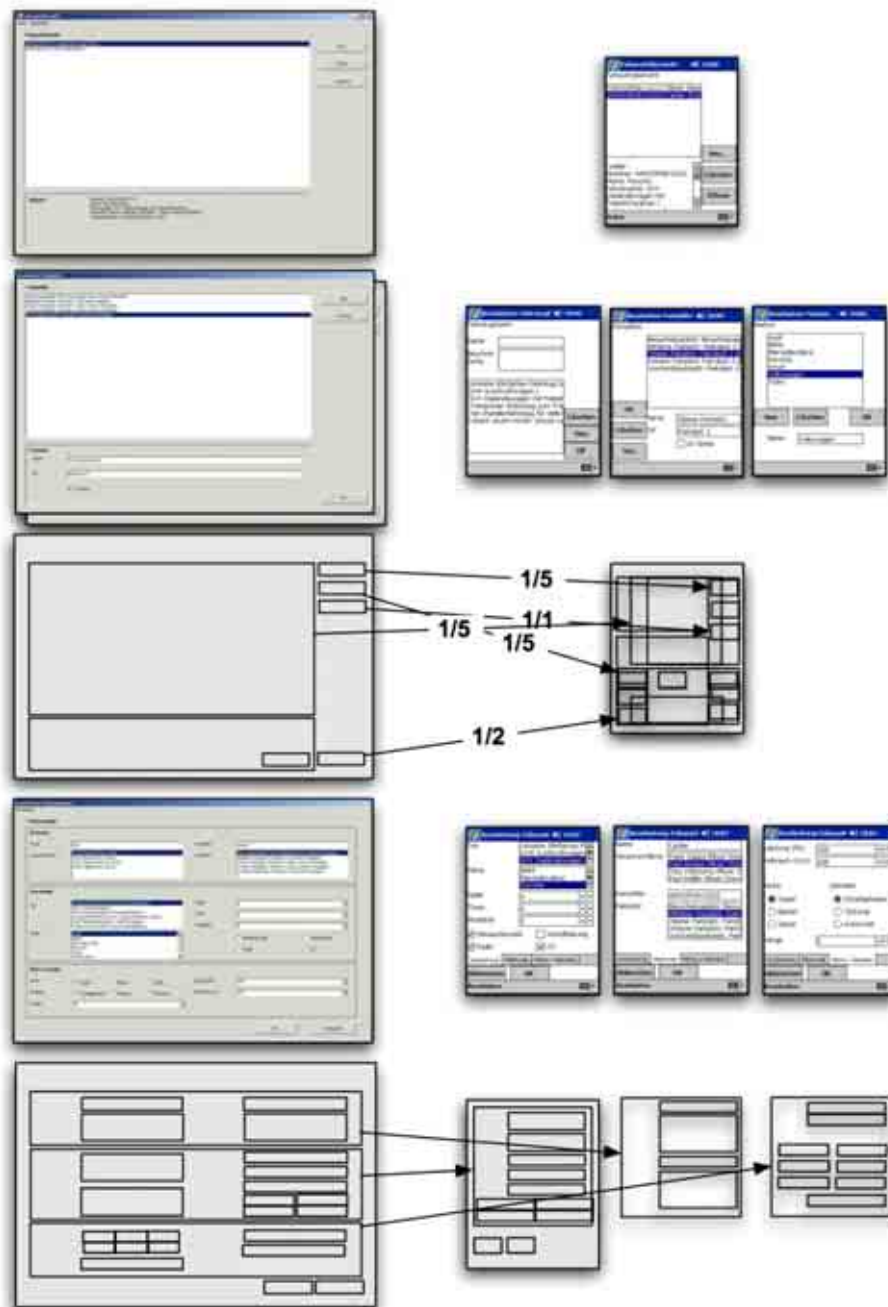


Abbildung 7.24: Hoch inkonsistente Abbildung

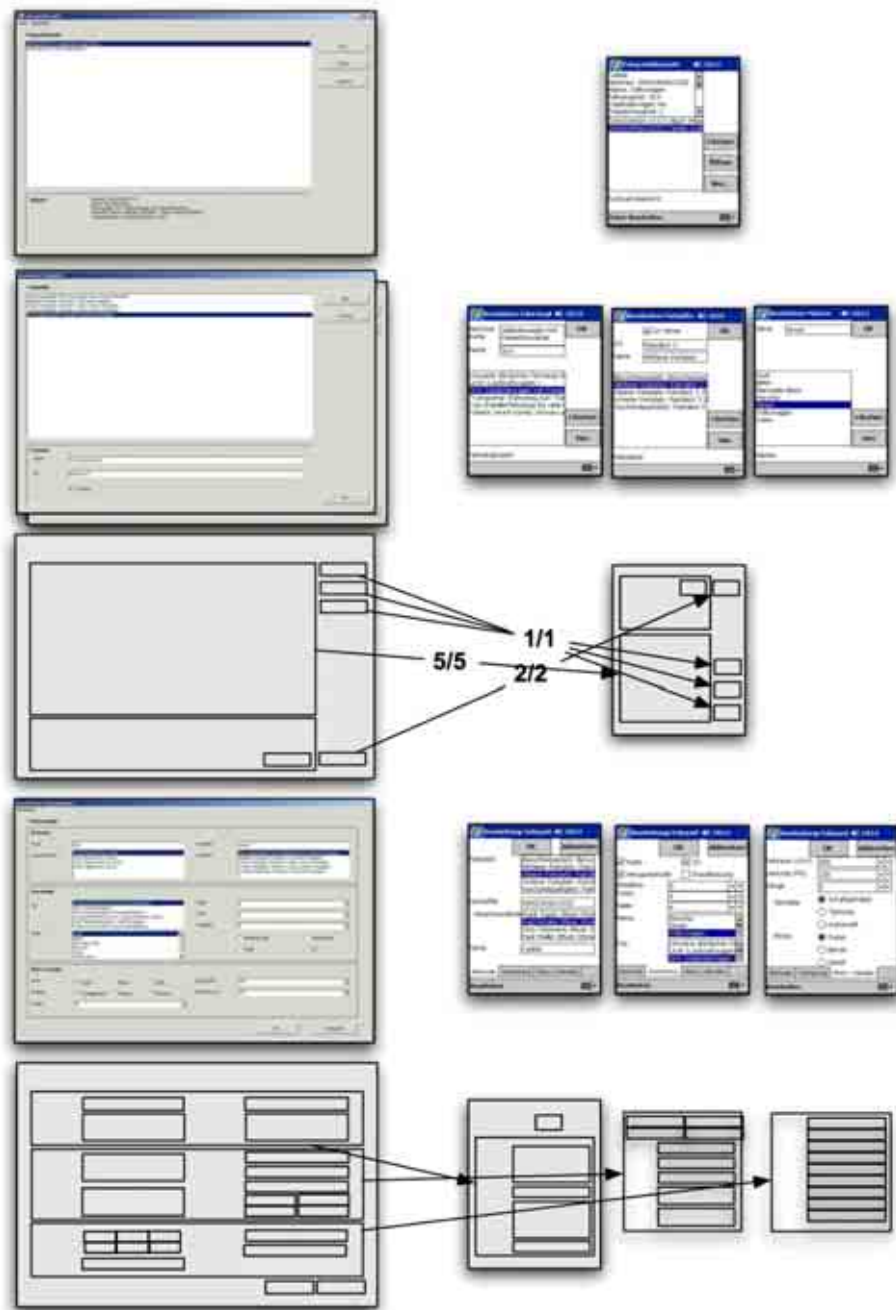


Abbildung 7.25: Konsistente gespiegelte Abbildung

ten Varianten. Ist jedoch die transformationale Konsistenz ausschlaggebend, dann sollte die Bearbeitungszeit für die gespiegelte Variante ähnlich wie die der konsistenten Variante sein ($(REG == MIR) < MED < HIG$).

Die Aussagen, die hier für die Bearbeitungszeiten getroffen wurden gelten in demselben Maße auch für die subjektive Bewertung der Benutzbarkeit.

Gesamtzeit Die gesamte Dauer (in System-Ticks, d.h. in Zeiteinheiten der CPU) der Bearbeitung wurde für jede Person aus den Log-Daten extrahiert.

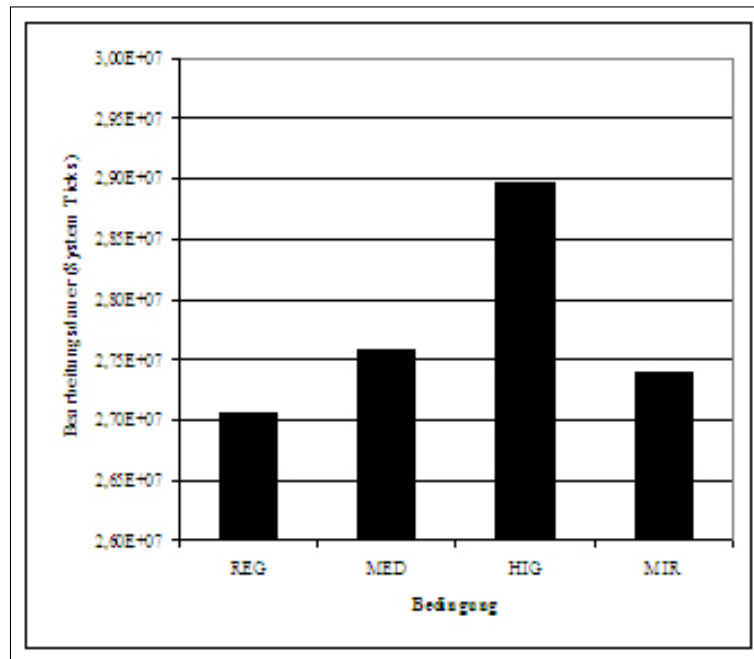


Abbildung 7.26: Mittlere Gesamt-Bearbeitungszeiten.

Abbildung 7.26 zeigt die mittleren Gesamt-Bearbeitungszeiten für die vier Gruppen. Der Trend zeigt die erwarteten Muster (REG: $\bar{x} = 27413734$, $s_{err} = 1611157$; MED: $\bar{x} = 27577343$, $s_{err} = 1801328$; HIG: $\bar{x} = 28970603$, $s_{err} = 2079995$; MIR: $\bar{x} = 27398195$, $s_{err} = 2079995$). Die Qualitative Hypothese, wie auch die Quantitative Hypothese lassen sich aus den Daten im Trend ableiten. Auch für die Gültigkeit der Transformationalen Hypothese gibt es im Trend Hinweise. Wie erwartet, erreicht allerdings keiner der Vergleiche ein signifikantes Niveau ($F(3, 14) = 0,14; p = .93$). Die hohe Varianz der Ergebnisse sowohl innerhalb des Verlauf einer individuellen Interaktion als auch zwischen Nutzern derselben Versuchsgruppe, lassen nur Trends ableiten.

Benutzbarkeit Die Benutzbarkeit der Anwendungen wurde jeweils direkt im Anschluss an deren Benutzung mittels SUS erhoben. Die geringere Wiederverwendbarkeit existierenden Wissens, so die Vermutung sollte sich bei dem Benutzer auch in der subjektiven Einschätzung der Benutzbarkeit niederschlagen.

Abbildung 7.27 zeigt die mittlere Veränderung der Benutzbarkeitseinschätzung zwischen Desktop-PC und Pocket PC für die vier Gruppen. Negative Werte zeigen eine schlechtere Bewertung der Pocket PC Anwendung als der Desktop PC Anwendung an. Obwohl alle Mittelwerte negativ sind, zeigen sie dasselbe Muster wie die Daten für die Bearbeitungszeiten und stützen damit die Transformationale Hypothese.

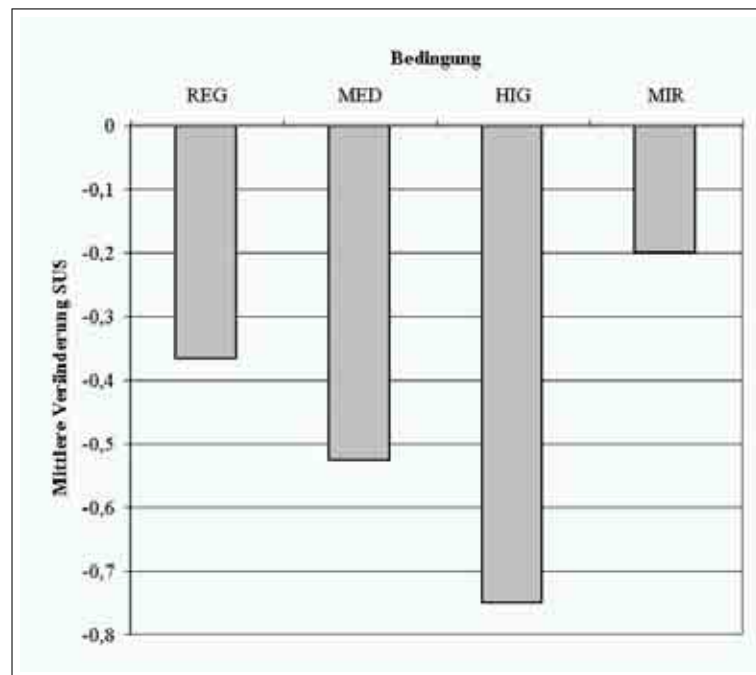


Abbildung 7.27: Mittlere Veränderung der subjektiven Benutzbarkeitseinschätzung.

7.3.3 Diskussion

Die Ergebnisse dieser Studie zur Validierung des Konsistenzmaßes bestätigen das transformationale Konsistenzmaß als auch die zugrundeliegende Transformationshypothese im Trend. Auch wenn der Versuchsaufbau, nicht dazu intendiert war eine statistische Absicherung der Hypothese zu erreichen, dient er dazu, eine der Kernthesen dieser Arbeit zu unterstützen. Die transformationale Konsistenz scheint sowohl die Bearbeitungsgeschwindigkeit als auch den subjektiven Eindruck der Benutzbarkeit eines Systems zu beeinflussen. Inkonsistente Abbildungen sind schlechter benutzbar als konsistente. Damit bestätigen diese Ergebnisse nochmals die Erkenntnisse aus den Studien in Abschnitt 7.2.

Wie diese Studie gezeigt hat, können Benutzer beim Wechsel zwischen verschiedenen Endgeräten von der konsistenten Gestaltung einer Anwendung profitieren. Dies wird auf den Transfer von Benutzungswissen zurückgeführt, der aufgrund der konsistenten Transformation der Darstellung erleichtert wird. Das mentale Anwendungsmodell des Benutzers, kann von diesem weiterverwendet werden, wenn für ihn konsistente Anpassungsregeln ableitbar sind. Inkonsistente Abbildungen führen zu einer geringeren Wiederverwendbarkeit und somit schlechterer Benutzbarkeit des plattformübergreifenden Systems. Das transformationale Konsistenzmaß ermöglicht eine quantitative Einschätzung der Benutzbarkeit eines Systems.

Im Gegensatz zu der klassischen Vorstellung von Konsistenz, die primär von einer gleichartigen Gestaltung ausgeht, erlaubt das transformationale Konsistenzmaß die Berücksichtigung plattform-spezifischer Optimierungen. Diese werden weiterhin als konsistent gewertet, wenn sie voraussagbar auf alle ähnlichen Elemente angewendet werden.

7.4 Zusammenfassung

In diesem Abschnitt wurden die wesentlichen Aussagen und Konzepte dieser Arbeit empirisch validiert.

Zunächst konnte gezeigt werden, dass die technische Umsetzung des plattformübergreifenden Darstellungssystems zu benutzbaren und auf benutzerspezifische Anforderungen (insbesondere behin-

derter Benutzer) anpassbaren User Interfaces führen. Die Validierung des Darstellungssystems wurde von Benutzern mit verschiedenen Einschränkungen erfolgreich getestet und ermöglichte eine angepasste multimodale Interaktion mit gängigen Informationssystemen. Nach der exemplarischen technischen Umsetzung, die in Abschnitt 6.1 beschrieben wurde, stand hier die empirische Evaluation des Systems im Vordergrund. Das System konnte hierbei erfolgreich getestet werden.

Die kognitionspsychologischen Grundlagen des transformationalen Konsistenzansatzes, wie er in dieser Arbeit entwickelt wurde, wurden im zweiten Teil dieses Abschnitts überprüft. Hier konnte gezeigt werden, dass die Annahme eines mentalen Anpassungsprozesses beim Wechsel zwischen verschiedenen Instanzen eines User Interface berechtigt ist. Dieser Prozess scheint auf eine visuelle mentale Repräsentation zuzugreifen auf die wiederum mentale Transformationsprozesse angewendet werden. Anhand der Replikation des klassischen Effekts der mentalen Rotation wurden Indizien für diese These aufgebracht.

Eine beispielhafte Anwendung einer regelbasierten Transformation in einer kommerziellen Kalenderapplikation konnte zeigen, dass neben der Steigerung der subjektiv wahrgenommenen Ähnlichkeit der Anwendungen zugleich eine Verbesserung der Benutzbarkeit möglich ist. Diese Erkenntnis ist insbesondere aus dem Grunde relevant, als dass Wege aufgezeigt wurden, wie diese Anpassung regelbasiert automatisch erfolgen kann. Somit wurde der Weg gewiesen, wie zugleich eine Beschleunigung als auch eine Verbesserung des plattformübergreifenden Entwicklungsprozesses möglich ist.

Das Konsistenzmaß, dessen Einsatz bei der Entwicklungsunterstützung in Abschnitt 6.2 diskutiert wird, wurde im letzten Teil dieses Abschnittes überprüft. Auch hier konnten die Annahmen der transformationalen Konsistenz erneut unterstützt werden. Transformationale Konsistenz, so erscheint es in Übereinstimmung mit der Transformationshypothese eignet sich zur Vorhersage des Wissenstransfers zwischen Plattformen und damit zur Verbesserung plattformübergreifender Anwendungen. Dies spricht für die Relevanz des Konsistenzmaßes in der Entwicklung plattformübergreifender Anwendungen. Weiterhin geben diese Ergebnisse einen Hinweis auf das Potential, welches in der Integration transformationaler Evaluierungsmechanismen in graphisch-interaktive Designumgebungen liegt.

8 Zusammenfassung und Ausblick

In diesem Abschnitt werden nochmals die wichtigsten Ergebnisse dieser Arbeit zusammengefasst. Zielstellung der Arbeit war es, Methoden zur Unterstützung bei der Entwicklung plattformübergreifender Benutzerschnittstellen zu entwickeln und zu realisieren. Im Rahmen dieser Arbeit wurden auf Basis psychologischer Konzepte Methoden entwickelt und getestet, die diese Unterstützung in den Entwicklungsprozess einbringen können.

Zunächst werden die Inhalte und Ergebnisse dieser nochmals zusammengefasst und mit den Kernzielen verglichen. Dann wird der wissenschaftliche Beitrag dieser Arbeit für die Forschungsgemeinschaft und die daraus entstehenden Verbesserungen bei der Entwicklung plattformübergreifender Benutzerschnittstellen angeführt. Zuletzt soll ein Ausblick Fragen herausstellen, die im Rahmen dieser Arbeit nicht beantwortet werden konnten oder erst entstanden.

8.1 Zusammenfassung

Das Problem, welches sich heute bei der Benutzung derselben Anwendung auf verschiedenen Endgeräten stellt, liegt im wesentlichen in der Tatsache, dass der Benutzer sich für jedes Gerät in der Regel auf eine deutlich unterschiedliche Anwendung einstellen muss. Der Transfer von Anwendungswissen zwischen Plattformen kann meist nur eingeschränkt stattfinden. Mit anderen Worten: kann ich eine Anwendung (z.B. eine Kalendersoftware) auf dem Desktop PC bedienen und habe mich an diese gewöhnt, bedeutet dies nicht, dass ich dieselbe Anwendung auf einem Pocket PC ebenfalls kenne und bedienen kann. Ziel der plattformübergreifenden Entwicklung ist es, die plattformübergreifende Benutzbarkeit zu verbessern, so dass der Bruch zwischen den Plattformen minimiert und der redundante Entwicklungsaufwand für den Hersteller reduziert wird.

Bei der Konzeption dieser Arbeit wurde Wert darauf gelegt, eine möglichst benutzerzentrierte Sichtweise einzunehmen, da der Mensch als der zentrale Akteur, zum einen in der Rolle als Entwickler, zum anderen als Anwender der realisierten Software, in einem Informationssystem stets im Vordergrund steht.

8.1.1 Inhalte

Die Forschungsarbeiten die im Rahmen dieser Arbeit durchgeführt wurden gliedern sich in die vier klassischen Phasen der wissenschaftlichen Forschung: *Aufnahme des Ist-Zustandes und Erarbeitung der Grundlagen*, eine konzeptionelle *Ausarbeitung der Lösungsansätze*, die prototypische *Umsetzung und Konkretisierung* der Konzepte und schlussendlich die *Bewertung* der Validität dieser Konzepte anhand empirischer Untersuchungen.

Aufnahme des Ist-Zustandes und Erarbeitung der Grundlagen In einer *initialen Benutzerstudie* wurde zunächst versucht, die kognitiven Prozesse zu erforschen, welche bei dem Wechsel zwischen Plattformen beim Nutzer auftreten. Die grundlegende Annahme dieser Arbeit sollte damit zunächst einer empirischen Überprüfung unterzogen werden. Als zentrale Hypothese wurde auf die Bedeutung der mentalen Repräsentation, des sog. *mental Modells* des Benutzers eingegangen und der Einfluss des Plattformwechsels auf diese mentale „Arbeitshypothese“ des Benutzers untersucht. Die Studie gibt Hinweise darauf, dass bei einem Wechsel zwischen Endgeräten existierendes Anwendungswissen des Benutzers über die Plattformen transferiert und angewandt wird. Je kompatibler dieses Wissen zwischen den Anwendungen, mit anderen Worten, je *konsistenter* die Anwendung auf den verschiedenen Plattformen, desto einfacher fällt dem Benutzer der Wechsel. Die beim Wechsel entstandenen Fehler wurden mittels der Fehlerklassifikation von Reason klassifiziert und Rückschlüsse auf die zugrundeliegenden mentalen Prozesse gezogen.

Basierend auf den Erkenntnissen dieser ersten Studie wurde anhand der existierenden Literatur eine Liste an *Anforderungen an die plattformübergreifende Entwicklung von Benutzerschnittstellen* gesammelt. Diese Anforderungen gliederten sich in benutzungsorientierte Anforderungen, wobei die Forderung nach einer plattformübergreifenden Konsistenz der Darstellung neben den Anforderungen an die Benutzbarkeit und der universellen Zugänglichkeit im Vordergrund stand. Bei der Zusammenstellung der Anforderungen an den Entwicklungsprozess wurde primär von den kritischen Thesen Trættebergs [Tra04] zu den User Interface Management Systemen (UIMS) und dessen Anwendungen der Kriterien des *Diffusion of Innovations* Modells von Rogers [Rog95, Rog97] ausgegangen, welche primär die Ursachen der mangelnden Akzeptanz von UIMS in der kommerziellen Entwicklung beleuchtet. Die Kriterien des Nutzens, der Kompatibilität, der Komplexität, der Testbarkeit und der Sichtbarkeit wurden hierbei beleuchtet und im Kontext der plattformübergreifenden Entwicklung diskutiert. Als wesentliche Problempunkte wurde dabei die Reduzierung der Komplexität und die Kompatibilität zu existierenden Entwicklungsmethoden in die Hauptziele dieser Arbeit aufgenommen. Die technischen Anforderungen an den Entwicklungsprozess sind im wesentlichen durch die konkrete Umsetzung determiniert und wurden daher sehr allgemein formuliert. Neben den üblichen Anforderungen an moderne Softwaresysteme wie die Plattformunabhängigkeit, Skalierbarkeit und Erweiterbarkeit stand hier die Verwendung von Standards als wesentlicher Aspekt im Vordergrund. Angesichts einer Vielzahl existierender akademischer Ansätze sollte keine weitere Beschreibungssprache für Benutzerschnittstellen entwickelt sondern auf existierende Standards aufgebaut werden.

Die *wissenschaftlichen Grundlagen* dieser Arbeit beziehen sich im wesentlichen auf die Komplexe des Wissenstransfers, damit verbunden das Thema der mentalen Repräsentationen und schließlich die Problematik der Konsistenz als wichtigem Kriterium bei der Gestaltung benutzbarer User Interfaces. Der Wechsel zwischen Endgeräten kann insbesondere dadurch erleichtert werden, dass der Benutzer vorhandenes Wissen auf dem neuen Gerät weiterverwenden und zusätzlich ausbauen kann. Die Zusammenfassung des Standes der Forschung auf dem Gebiet des Lernens und Wissenstransfers gibt einen Überblick und lenkt die Aufmerksamkeit darauf, wie dieses Wissen repräsentiert ist. Einer der Ansätze postuliert das Vorhandensein gewisser *analoger* Repräsentationen in der Vorstellung des Benutzers. Dieses Konzept der mentalen Modelle fand in der Forschung zur Mensch-Maschine Interaktion große Resonanz und wurde hier ausführlich beschrieben. Wiederum aufbauend hierauf wurde argumentiert, dass die Konsistenz der Abbildung den Transfer von Wissen wesentlich beeinflussen kann, und somit notwendig für die effiziente Gestaltung benutzbarer plattformübergreifender Schnittstellen ist.

Die Vorstellung des *Standes der Technik* auf dem Gebiet der plattformübergreifenden Entwicklung von Benutzerschnittstellen zeigte, dass derzeit insbesondere modellbasierte UIMS verschiedener Komplexität (d.h. mit verschiedenen Abstraktions- und Modellierungsebenen) entwickelt werden. Als wichtigste Ansätze wurden hierbei die *User Interface Markup Language (UIML)* [AP99], das USIXML-System [VLM⁺04], TERESA [BCMP04] und XIIML [PE04] aufgeführt und verglichen. Der Vergleich dieser Ansätze mit den Anforderungen konnte verdeutlichen, dass insbesondere die Sicherstellung der Benutzbarkeit und die Kompatibilität zu existierenden Entwicklungsmethoden nicht ausreichen unterstützt werden. Modellbasierte Systeme bieten zwar die besten Potentiale der plattformübergreifenden Anpassung, sie erfordern jedoch gleichzeitig einen enormen Einarbeitungsaufwand und die bedingungslose Umstellung der Entwicklungsmethodik. Bezüglich der automatischen Evaluation und Unterstützung benutzbarer User Interfaces konnte zum einen in den Grundlagen allgemein, und im Stand der Technik speziell für die Messung der Konsistenz gezeigt werden, dass insbesondere die Integration solcher Methoden in den Entwicklungsprozess selbst, derzeit noch kaum unterstützt wird. Insbesondere die Einbettung von leitenden und assistierenden Funktionen bei der graphisch-interaktiven Gestaltung ist bislang kaum umgesetzt.

Ausarbeitung der Lösungsansätze Die Konzepte, welche in dieser Arbeit entwickelt wurden richten sich nach den Anforderungen aus, welche im ersten Teil der Arbeit entwickelt wurden.

Des Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen hat zum Ziel, die benutzerzentrierte Herangehensweise des *Usability Engineering Lifecycles* [May99] mit den Anforderungen der Kompatibilität zur existierenden Entwicklungsmethoden und niedrigen

Komplexität zu vereinen und auf die plattformübergreifende Entwicklung anzuwenden. Im wesentlichen wurden hierbei drei zentrale Konzepte entwickelt, die im weiteren Verlauf der Arbeit verfeinert und angewandt wurden.

- Die *Sequentielle Anpassung* der abstrakten Darstellung an die plattformspezifischen Eigenschaften sollte die kontrollierte Modifikation der Darstellung sowohl *bottom-up* als auch *top-down* ermöglichen. Das bedeutet, dass zum einen von der Anwendungsseite ausreichend Kontrolle über die gestalterischen Aspekte eines UI vorhanden ist, diese aber nicht notwendig in voller Komplexität durchscheinen. Zum anderen ist die Möglichkeit der graphisch-interaktiven Gestaltung in sog. *Visual Designern* gegeben, wobei das *Mapping Problem* durch die Isolierung der betroffenen Ebene (Standard oder Konkretes Design) umgangen werden kann. Multimodale und kontextangepasste Darstellungsmöglichkeiten wurden in diesem Umfeld ebenfalls besprochen.
- *Geordnete Entwicklungspfade* sollen ebenfalls dazu dienen, die Problematik des *top-down* Zurückführens von Änderungen in der konkreten Darstellung in die abstrakteren Ebenen der Interfacespezifikation (*Mapping*) zu kontrollieren. Weiterhin ist dieses Konzept geeignet die *bottom-up* Entwicklung zu vereinfachen. Indem der Entwickler primär für eine Zielplattform entwickelt, ist die Entwicklung für eine konkrete Darstellung möglich. Der Entwickler ist somit nicht gezwungen in abstrakten Dimensionen zu denken. Das System passt diese dann transparent auf andere plattformspezifische Darstellungen an.
- Das *Transformationsparadigma* berücksichtigt das Konzept der geordneten Entwicklungspfade und leitet daraus die aus der Diskussion der Grundlagen kognitionspsychologisch begründete These ab, dass die Abbildung zwischen Endgeräten vom Benutzer als Transformation wahrgenommen wird. Transformationen lassen sich in elementare Transformationsschritte zerlegen. Diese Transformation ermöglicht es das existierende Anwendungswissen auf das neue Endgerät zu übertragen. Ist die Transformation über alle Dialoge einer Anwendung konsistent angewandt, fällt der Transfer leichter.

Insbesondere die Hypothese von der Transformation der Darstellung wurde genutzt um daraus die elementaren *Transformationsstrategien* abzuleiten, welche bei der Abbildung angewandt werden können. Ein Klassifikationsschema für diese Transformationsstrategien wurde erarbeitet und mit einer Liste graphischer Transformationen gefüllt. Durch die Identifizierung dieser Strategien kann bei der Entwicklung von User Interfaces gezielt auf die konsistente Verwendung dieser Strategien geachtet werden und die Anpassung hinsichtlich der Konsistenz optimiert werden.

Die wichtigste Ausprägung des Transformationsparadigmas in dieser Arbeit stellt jedoch die Ableitung des *transformationalen Konsistenzmaßes* dar. Dieses Maß ermöglicht auf generische Weise die Bestimmung der Konsistenz der Beziehung zweier Gruppen von Darstellungen bezüglich eines Merkmales. Es gibt hierbei zwei Berechnungsgrundlagen, zum einen der stochastische, zum anderen der varianzanalytische Ansatz. Im wesentlichen drückt dieses Maß die Konsistenz in Form der bedingten Wahrscheinlichkeit einer Merkmalsausprägung in der zweiten Gruppe aufgrund der Merkmalsausprägung in der ersten Gruppe aus.

Umsetzung und Konkretisierung Die Umsetzung der Konzepte in konkrete Anwendungen und Prototypen erfolgte zum einen in Form eines plattformübergreifenden Darstellungssystems für Desktop, Pocket PC und Smartphone, zum anderen wurden Ansätze zur Entwicklungsunterstützung konsistenten plattformübergreifenden Designs dargestellt und zum Teil umgesetzt.

Das *Darstellungssystem* hatte in erster Linie die Funktion den Abstraktionskern auf Basis des W3C Standards XForms und der Cascading Style Sheets (CSS) zu realisieren. Dieser Abstraktionskern wurde zur Anwendungsseite hin durch zwei Programmierungs-Schnittstellen verschiedener Komplexität gekapselt. Es sollte damit gezeigt werden, dass durch die Kapselung der abstrakten Darstellung hinter einer gewöhnlichen Toolkit-API die Komplexität des Entwicklungsprozesses reduziert und die Kompatibilität zu existierenden Entwicklungsmethoden verbessert werden kann. Eine komplexere API ermöglichte zugleich die Anbindung der Darstellung an andere UIMS und die Verwendung aller Funktionen des Abstraktionskerns. Die Problematik der Darstellungsanpassung wurde durch das Konzept der sequentiellen Anpassung gelöst, indem die Darstellung für die verschiedenen

Plattformen zunächst standardmäßig und dann auf die speziellen Designanforderungen oder Kontextparameter angepasst werden konnte. Über die modulare Abtrennung der plattformspezifischen Darstellung sind zum einen multimodale Ausgaben, zum anderen die physikalische Abtrennung von Anwendung und Darstellung möglich.

Methoden zur Unterstützung der Entwicklung konsistenter Darstellungen, sowohl innerhalb einer Anwendung auf einem Endgerät als auch plattformübergreifender Anwendungen wurden entwickelt. Diese sollten in erster Linie so gestaltet sein, dass sie sich in der graphisch-interaktiven Gestaltungsprozess einbinden und durch visuellen Rückmeldungen direkt zu einer Unterstützung des Designprozesses genutzt werden können. Die gängigen summativen Ansätze, wie die in den existierenden Evaluierungswerkzeugen eingebracht wurden haben den Nachteil, dass sie orthogonal zu dem eigentlichen Prozess stehen. Die Einbindung des Konsistenzmasses in einer anwendungsweiten Perspektive, die Anwendung von Transformationsstrategien und die Bereitstellung von Transformationsmustern wurden insbesondere für die Unterstützung der plattformübergreifenden Konsistenz vorgeschlagen.

Bewertung Die Bewertung der hier erarbeiteten Konzepte erfolgte aufgrund der benutzerzentrierten Orientierung dieser Ansätze in empirischer Weise aufgrund von Benutzungsstudien. Nachdem die technische Umsetzbarkeit anhand prototypischer Systeme erwiesen war, musste die Validität der zugrundeliegenden Annahmen überprüft werden. Diese Überprüfung erfolgte in drei Schritten.

Zunächst wurde das Darstellungssystem auf seine Benutzbarkeit überprüft. Die Annahme der universellen Zugänglichkeit und der Benutzbarkeit des Darstellungssystem in multimodalen Anwendungsszenarien wurden in drei Studien mit körperlich eingeschränkten Nutzern getestet. Die Benutzbarkeit des Systems wurde belegt.

In einem zweiten Satz von Untersuchungen wurde die Validität der Transformationshypothese anhand der Paradigmas der mentalen Rotation validiert. Es konnte gezeigt werden dass beim Wechsel zwischen zwei zueinander rotierten Darstellungen derselben Oberfläche ein Effekt des Rotationswinkels auf die Reaktionszeit zu beobachten ist. Dieser Effekt stützt die Hypothese dass eine Transformation (in diesem Fall Rotation) auf der Ebene der mentalen Repräsentation stattfindet. Weiterhin wurde eine regelbasierte Transformation bei der Anpassung einer kommerziellen Anwendung auf ein Mobilgerät angewendet und erwies sich als besser benutzbar als die verfügbare Originalanwendung.

Die dritte Studie diente schließlich dazu, die Validität des transformationalen Konsistenzmaßes zu überprüfen. Es konnte nicht nur gezeigt werden, dass das Konsistenzmaß qualitative Vorhersagen über die Benutzbarkeit einer abgeleiteten Anwendungsoberfläche ermöglicht, sondern dass qualitative Vorhersagen sowohl in der Dimension der Bearbeitungszeit als auch in der subjektiven Einschätzung der Benutzbarkeit möglich sind. Diese Studie konnte weiterhin einen zusätzlichen Beleg für die Transformationshypothese liefern, da gezeigt werden konnte, dass eine gespiegelte Abbildung genauso schnell zu bedienen ist wie eine im klassischen Sinne konsistente Abbildung, diese Abbildung also gleichermaßen als konsistent wahrgenommen wird. Damit konnte der klassische Konsistenzbegriff zugunsten der der transformationalen Konsistenz widerlegt werden.

8.1.2 Ziele

Zu Beginn dieser Arbeit wurden aus der Problemstellung heraus *drei Kernziele* definiert. Diese Ziele sollten im Rahmen dieser Arbeit erreicht werden oder Beiträge zu deren Erreichung entwickelt werden. In diesem Abschnitt werden nochmals die wesentlichen Ergebnisse im Kontext dieser Ziele aufgeführt.

Sicherstellung der Benutzbarkeit *Der Entwickler erhält Unterstützung bei der Gestaltung benutzbarer plattformübergreifender Benutzerschnittstellen. Die Entwicklungswerkzeuge sollen den Gestaltungsprozess leiten und existierendes Wissen über Voraussetzungen benutzbarer Systeme, wie z. B. plattformübergreifende Konsistenz, in den Prozess einfließen lassen. Dies soll in Form von automatisierten Methoden erfolgen.*

Das transformationale Konsistenzmaß kann, eingebunden als Unterstützung in graphische Entwicklungsumgebungen dazu genutzt werden, die *Benutzbarkeit* oder auch Kontinuität plattformübergreifender Anwendungen zu verbessern. Für den Benutzer ergibt sich daraus der Vorteil, Anwendungen einfacher bedienen zu können ohne für jedes Endgerät neue Strategien entwickeln zu müssen.

Der Entwickler kann anhand der *Transformationsstrategien* und mit Hilfe des *Konsistenzmaßes* bereits während des Entwicklungsprozesses die spätere Wiedererkennbarkeit des User Interface auf verschiedenen Plattformen bewerten und bei der Gestaltung zwischen plattform-spezifischer Optimierung und plattformübergreifender Konsistenz abwägen.

Reduzierung des Anpassungsaufwandes *Es sind Methoden notwendig Anwendungsoberflächen einfach und schnell an neue Geräte anzupassen. Diese Methoden sollten möglichst transparent für den Entwickler automatisch oder unterstützt innerhalb des Darstellungssystems angewandt werden. Die Möglichkeit der Wiederverwendung existierender Lösung in Form von Design Patterns stellt eine weitere Lösung dar.*

Der transformationale Entwicklungsansatz als zentraler Beitrag dieser Arbeit kann bei der Entwicklung plattformübergreifender Benutzerschnittstellen—als einer der Herausforderungen der Zukunft—zu einer Beschleunigung der Prozesse durch Unterstützung und Automatisierung führen. Als *ökonomischer Vorteil* ergibt sich hieraus eine klare Kostenersparnis, schnellere Produktionszeiten und flexiblere Reaktion auf Markterfordernisse für die Hersteller plattformübergreifender Anwendungen.

Reduzierung des Einarbeitungsaufwandes *Die Entwickler von plattformübergreifenden Anwendungen sollen—soweit dies möglich ist—dieselben Entwicklungsmethoden verwenden können wie bislang. Dies bedeutet das der Einsatz von graphischen Entwicklungswerkzeugen oder erfahrungskonformen Anwendungsprogrammierungsschnittstellen (APIs) unterstützt werden sollen.*

Die Abstraktion von Darstellung und Anwendungslogik in dem hier beschriebenen Darstellungssystem, sowie die generische Definition des Konsistenzmaßes können dazu genutzt werden modalitätsunabhängige, multi- oder polymodale User Interfaces zu entwickeln und in konsistenter Form darzustellen. Wie gezeigt wurde ist dies ein weiterer Beitrag für die Vision der *universell benutzbaren* und *universell zugänglichen* Softwaresysteme der Zukunft.

Zusammenfassend lässt sich feststellen, dass alle drei Kernzielen erreicht wurden. Der transformationale Entwicklungsansatz und die daraufaufbauenden Methoden unterstützt die einfache und schnelle Entwicklung benutzbarer Benutzerschnittstellen für plattformübergreifende Anwendungen.

8.2 Wissenschaftlicher Beitrag

Der wesentliche Beitrag dieser Arbeit besteht in der Anwendung kognitionspsychologischer Erkenntnisse bei der Konzeption, Umsetzung und Validierung eines transformationalen Ansatzes der Entwicklung plattformübergreifender Benutzerschnittstellen. Hierauf aufbauend wurde eine Entwicklungsmethodik formuliert und in einem System prototypisch umgesetzt. Ein Konsistenzmaß wurde abgeleitet, welches die *quantitative* Bestimmung der plattformübergreifenden Konsistenz als einem wesentlichen Aspekt der Benutzbarkeit von User Interfaces ermöglicht.

Eingebettet in ein *Referenzmodell der plattformübergreifenden Entwicklung von Benutzerschnittstellen* wurden folgende Unterstützungsmethoden erarbeitet:

Plattformübergreifende Entwicklungsplattform — Das hier vorgestellte Entwicklungsmodell betont die Notwendigkeit der Integration plattformübergreifender Methoden in den heute üblichen Entwicklungsprozess und damit die Unterstützung einfacher API- und Toolkit-basierter Programmierung sowie der graphisch-interaktiven Gestaltung in visuellen Bearbeitungswerkzeugen. Eine mehrstufige Entwicklungsschnittstelle wurde konzipiert und umgesetzt, welche die Komplexität des zugrundeliegenden Systems kapselt. Im Rahmen eines Darstellungssystems für verschiedene Plattformen, wurde das Prinzip der hierarchischen Pfade

umgesetzt. Das toolkit-basierte auf dem W3C aufbauende plattformübergreifende Darstellungssystem ermöglicht die transparente Entwicklung graphischer Oberflächen für verschiedene Endgeräte.

Transformationsstrategien — Die Transformationshypothese betrachtet die plattformübergreifende Abbildung einer Anwendung als eine Transformation. Diese setzt sich aus elementaren Transformationsschritten zusammen. Dieses Konzept beruft sich auf kognitionspsychologische Erkenntnisse zur menschlichen Wahrnehmung und Informationsverarbeitung. Dieser transformationale Ansatz eröffnet aus akademischer Sicht eine neue Perspektive auf die Problematik der plattformübergreifenden Entwicklung. Zum anderen entsteht hieraus aus technischer Sicht die Möglichkeit neuer Methoden der Entwicklungsunterstützung. Die Anwendung von Transformationsstrategien, so konnte gezeigt werden, führt zu einer konsistenteren und damit besser benutzbaren Gestaltung plattformübergreifender User Interfaces.

Plattformübergreifendes Konsistenzmaß — Eine Methode zur Messung der plattformübergreifenden Konsistenz von Benutzerschnittstellen wurde basierend auf der erarbeiteten Transformationshypothese entwickelt und validiert. Die Einbindung dieses Maßes in den graphisch-interaktiven Gestaltungsprozess von User Interfaces wurde anhand verschiedener Konzepte und Prototypen exemplarisch vorgeführt. Das Potential dieser Methoden für die Beschleunigung des Entwicklungsprozesses und die Verbesserung der Benutzbarkeit plattformübergreifender Systeme wurde empirisch belegt.

Die Konzepte dieser Arbeit wurden empirisch in mehreren Studien überprüft. Es konnte gezeigt werden, dass der Benutzer eine räumlich-graphischen mentalen Repräsentation der Benutzungsoberfläche beim Übergang zwischen Endgeräten einsetzt um sein Wissen über die Gestaltung der Oberfläche zur Lokalisierung von Funktionen anzupassen und weiter nutzen zu können. Die Replizierung des klassischen Effektes der mentalen Rotation [CS73] kann als Indiz für diese Hypothese gewertet werden. Diese Argumentationskette wurde weitergeführt um eine Operationalisierung der Konsistenz abzuleiten, welche im Kontrast zu der klassischen Konsistenzkonzepten von einer transformationalen Perspektive ausgeht. Dadurch können auch grundlegend verschiedene Darstellungen zueinander in Beziehung gesetzt werden, wie dies für die Evaluierung von plattformübergreifenden User Interfaces notwendig ist. Die Abbildung zwischen Geräten muss folglich nicht möglichst ähnlich sein, sondern einer möglichst gleichmäßigen Transformation entsprechen.

8.3 Ausblick

Die in dieser Arbeit entwickelten Ansätze haben eine neue Perspektive auf die Gestaltung und Bewertung plattformübergreifender Benutzerschnittstellen eröffnet. Diese birgt neue Möglichkeiten für die praktische Umsetzung der Entwicklungsunterstützung, eröffnet jedoch auch neue Forschungsfragen zu den kognitiven Prozessen in der Mensch-Maschine Interaktion.

Die praktische *Umsetzung* der Entwicklungsunterstützung sollte in erster Linie in Form einer Integration in existierende Werkzeuge der graphisch-interaktiven Spezifikation erfolgen. Der transformationale Ansatz betont die Bedeutung der Konsistenz der Abbildung zwischen plattformspezifischen Darstellungen und öffnet mit dem transformationalen Konsistenzmaß die Möglichkeit diese durch geeignete Werkzeuge zu unterstützen.

Ausgehend von den hier vorgestellten Methoden zur Unterstützung ist die Umsetzung und Weiterentwicklung dieser Konzepte in existierenden toolkitbasierten Entwicklungsumgebungen wie dem *Visual Studio .Net* von Microsoft oder der *Eclipse-SDK* eine der größten Herausforderungen für die unmittelbare Zukunft. Nachdem diese Arbeit die Validität der transformationalen Ansatzes belegt hat und die Relevanz der transformationalen Konsistenz für die plattformübergreifende Benutzbarkeit anhand empirischer Methoden aufzeigte, kann nun die Umsetzung und Integration dieser Methoden in existierenden Werkzeugen den wissenschaftlichen Beitrag in den effektiven Nutzen für die Softwareentwicklung umgewandelt werden. Die hier beschriebenen Methoden müssen hierbei noch auf ihre Effektivität bei der Entwicklungsunterstützung überprüft und gegebenenfalls weiterentwickelt werden. Die Einbettung der Entwicklungsmethodik in einen modellbasierten Entwicklungsprozess und die dafür verfügbaren graphischen Werkzeuge wie z. B. GrafiXML [Van05] stellt eine weitere

mögliche Anwendung der hier vorgestellten Entwicklungsunterstützung an. Während toolkitbasierte Systeme hierfür zusätzliche anwendungsweite Informationen sammeln müssen, stehen diese in modellbasierten System meist schon zur Verfügung.

Eine Erweiterung des Umfangs des Darstellungssystems und des XForms-Standards mit Fokus auf primär direkt-manipulative Interaktionsmetaphern anstelle des derzeit fokussierten formbasierten Paradigmas könnte die Anwendbarkeit des Darstellungssystems auch auf komplexere Systeme ermöglichen. Diese ist derzeit in erster Linie durch den Umfang des Standards eingeschränkt. Im Rahmen der *Device Independency Workgroup* [Wor01] sowie der *Multimodal Interaction Workgroup* [Wor02] des W3C werden jedoch bereits Standards entwickelt, die aufbauend auf XForms, XHTML und anderen W3C Standards die plattformunabhängige Darstellung von User Interfaces zum Ziel haben. Die Verwendung dieser Standards in Erweiterung des existierenden XForms-Modells, welches derzeit den abstrakten Kern des Darstellungssystems bildet stellt ebenfalls ein Ziel der zukünftigen Verbesserung und Erweiterung der plattformübergreifenden Entwicklung dar.

Aus dem transformationalen Entwicklungsansatz ergeben sich eine Reihe von *Forschungsfragen*, die Gegenstand zukünftiger Arbeiten sein werden.

Die Frage welche Aspekte der Gestaltung im wesentlichen die Wahrnehmung der Konsistenz prägen muss geklärt werden um für die Integration des Konsistenzmaß bei der Einstellung der wesentlichen Parameter gezielt Unterstützung geben zu können. In der vorliegenden Arbeit wurde der Schwerpunkt auf die Position und Orientierung, sowie in geringerem Maße auch auf die Farbgestaltung und Typographie eingegangen. Abstraktionen von Gruppenkonstellationen, Formen, zeitlichen und interaktiven Aspekten könnten dazu dienen das Konsistenzmaß auch auf weitere Aspekte der Gestaltung auszuweiten.

Die Frage der kognitiven Gewichte verschiedener Transformationen wird in dieser Arbeit nur oberflächlich diskutiert. Es kann angenommen werden, dass einige Transformationen für den Benutzer wesentlich leichter nachzuvollziehen sind als andere. In der Systemvalidierung wurde ein experimentelles Paradigma vorgestellt bei dem der Effekt der Rotation erfasst wurde. Dieses Paradigma könnte nun auch auf weitere Transformationen angewendet werden, um so diese kognitive Last der einzelnen Transformationen, sowie deren Interaktion untereinander zu untersuchen. Zunächst kann hier ein einfacher additiver Effekt angenommen werden. Die Frage, wann der kognitive Aufwand der Transformation zu groß wird und der Benutzer beginnt eine neue Repräsentation aufzubauen passt sich ebenfalls in diesen Fragenkomplex ein.

In der einleitenden Studie wurde der Effekt des Wechsels zwischen Plattformen zunächst explorativ untersucht. Zur Analyse der Fehler wurde hierbei auf die Fehlerklassifikation von Reason [Rea90] zurückgegriffen. Die Art des Fehlers, so die Annahme, sollte Auskunft über dessen Ursprung geben, sei es als Übertragungsfehler von der vorherigen Plattform, sei es als regelbasierte Fehler auf der aktuellen Plattform. Dieser Zusammenhang wurde in erster Linie als Klassifikationsschema zur Einordnung verwendet. Obwohl die Gesamtheit der Ergebnisse, d. h. auch die Untersuchung der mentalen Modelle, ein konsistentes Bild ergaben und diese Annahme im Großen und Ganzen stützen, steht eine empirische Validierung dieser Annahme noch aus. Da diese Validierung über die Zielsetzung dieser Arbeit hinausging, konnte hierauf nicht weiter eingegangen werden. Dennoch scheint dieser Zusammenhang zur Bestimmung möglicher Kriterien bei der Evaluation plattformübergreifender Anwendungen eine Rolle spielen. Die Ableitung von Gestaltungshinweisen und Methoden könnte ein Resultat sein.

Eine weitere Methode, die an dieser Stelle nur unvollständig ausgearbeitet und validiert werden konnte ist die Methode der Graphischen Externalisierung. Diese wurde genutzt um ein graphisches Bild der mentalen Repräsentation des Nutzers von der Anwendung zu erhalten. Im Zusammenhang der Untersuchungen in dieser Arbeit wurde an verschiedener Stelle auch diese Methode angewendet. Die Ergebnisse konnten hier jedoch nicht aufgeführt werden ohne den Rahmen zu sprengen. Diese Methode scheint jedoch insbesondere bei der Untersuchung der mentalen Repräsentation von *naiven* Nutzern, bzw. Novizen genutzt werden können um die Kommunikation über die Anwendung zu leiten. Anwender mit technischer Vorbildung scheinen dazu zu tendieren, Metaphern wie Fluss- oder UML-Diagramme zu nutzen. Die einfache Anwendbarkeit dieser Methode wie ihr Potential bezüglich der Kommunikation komplexer Sachverhalte mit naiven Nutzern scheint es zu rechtfertigen,

weitere Untersuchungen zur systematischen Nutzung sowie zu nutzerspezifischen Unterschieden anzustellen.

Der transformationale Entwicklungsansatz und das daraus abgeleitete Konsistenzmaß wie sie in dieser Arbeit entwickelt und empirisch validiert wurden, stellen einen wichtigen Beitrag zur Verbesserung und Effektivierung der plattformübergreifenden Entwicklung und der Benutzbarkeit durchgängiger Anwendungen dar. Dieses Problemfeld gewinnt in der immer heterogeneren Landschaft der Informationstechnik und Endgeräte an Bedeutung und wird mit der Verfügbarkeit multimodaler Systeme in naher Zukunft zusätzlich an Komplexität zunehmen. Aufgrund des generischen Ansatzes der transformationalen Entwicklungsmethodik kann diese auch in der modalitätsübergreifenden Abbildung eingesetzt werden und dort den Entwickler bei der Gestaltung benutzbarer Systeme unterstützen. Der so entstandene Gewinn an Flexibilität bei der Anpassung an Modalitäten und Endgeräte kann genutzt werden um dem Ideal der Universellen Nutzbarkeit und Zugänglichkeit ein Stück näher zu kommen.

Am Anfang dieser Arbeit stand die Frage Ben Shneiderman's [Shn03] nach der Entwicklung plattformübergreifender Anwendungen, die auf jedem Gerät in gleicher Weise einfach zu bedienen sind. Die Konzepte und Lösungsansätze, die hier entwickelt wurden können dazu beitragen dieses Ziel näher zu bringen:

„Shouldn't software be designed so that users could run the same calendar program on a palm-sized device, a laptop, and a wall sized display?“

Literaturverzeichnis

- [ABDH89] ABOWD, Gregory ; BOWEN, Jonathan ; DIX, Alan ; HARRISON, Roger: User Interface Languages: A Survey of Existing Methods / Programming Research Group, Oxford University. Oxford, UK, 1989 (PRG-TR-5-89). – Technical Report
- [AD67] ANNETT, J. ; DUNCAN, K.: Task analysis and training in design. In: *Occupational Psychology* 41 (1967), S. 211–221
- [Agr86] AGRESTI, W. W.: What are the New Paradigms? In: AGRESTI, W. W. (Hrsg.): *New Paradigms for Software Development*. Washington D.C., USA : IEEE Computer Society, 1986
- [AHB⁺98] ARTIM, John ; VAN HARMELEN, Mark ; BUTLER, Keith ; GULIKSEN, Austin ; KOVACEVIC, Srdjan ; LU, Shijian ; OVERMEYER, Scott ; REAUX, Ray ; ROBERTS, Dave ; TARBY, Jean-Claude ; VAN DERLINDEN, Keith: Incorporating work, process and task analysis into commercial and industrial object-oriented systems development / Centre for User Oriented IT Design, Royal Institute of Technology. Stockholm, Sweden, October 1998 (CID-99). – Forschungsbericht
- [AIS⁺77] ALEXANDER, Christopher ; ISHIKAWA SARA ; SILVERSTEIN, Murray ; JACOBSON, Max ; FIKSDAHL-KING, Ingrid ; ANGEL, Schlomo: *A pattern language: towns, buildings, construction*. New York, USA : Oxford University Press, 1977
- [AKSA03] ABRAN, Alain ; KHELIFI, Adel ; SURYN, Witold ; AHMED, Seffah: Consolidating the ISO Usability Models. In: *Proceedings of the 11th International Software Quality Management Conference*. Glasgow, UK, April 23-25 2003
- [Ale79] ALEXANDER, Christopher: *The timeless way of building*. Oxford, UK : Oxford University Press, 1979
- [And93] ANDERSON, John R.: *Rules of the mind*. Hillsdale, NJ : Lawrence Erlbaum, 1993
- [And95] ANDERSON, John R.: *Cognitive psychology and its implications*. 4th. New York, USA : W. H. Freeman, 1995
- [And96] ANDERSON, John R.: ACT — A simple theory of complex cognition. In: *American Psychologist* 51 (1996), April, Nr. 4, S. 355–365
- [AP99] ABRAMS, Marc ; PHANOURIU, Constantinos: UIML: An XML language for building device-independent user interfaces. In: *Proceedings of XML'99*. Philadelphia, USA, 1999
- [App87] APPLE COMPUTER, INC: *HyperCard User's Guide*. Apple Computer, 1987
- [App92] APPLE COMPUTER INC.: *Macintosh Human Interface Guidelines*. Reading, MA : Addison-Wesley, 1992
- [App97] APPLE COMPUTER, INC.: *Apple Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley, 1997
- [App00] APPLETON, Brad. *Patterns and Software: Essential Concepts and Terminology*. <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>. (retrieved: 27.01.2006) 2000
- [App03] APPLE COMPUTER, INC. *Apple Human Interface Guidelines*. Webpage. October 2003
- [App05] APPLE COMPUTER, INC. *Interface Builder 2.4*. Software is part of Apple XCode Developer Tools (Mac OS X 10.4). 2005
- [ARB06] ALEXANDERSSON, Jan ; RICHTER, Kai ; BECKER, Stephanie: I2HOME: Benutzerzentrierte Entwicklung einer offenen standardbasierten Smart Home Plattform. In: *Proceedings of USEWARE 2006*. Düsseldorf, Germany : VDI, October 10.11 2006

- [Arn82] ARNHEIM, Rudolf: *Die Macht der Mitte: Kompositionslehre für die bildenden Künste*. Deutsche Übersetzung, 1983. Köln : DuMont, 1982
- [ARZ⁺05] ALEXANDERSSON, Jan ; RICHTER, Kai ; ZIMMERMANN, Gottfried ; WEGGE, Klaus-Peter ; SLAVIK, Pavel ; PALONC, Celine ; JÜRGEN, Bund. *Intuitive Interaction for Everyone with Home Appliances based on IndustryStandards: I2HOME*. Project Proposal. September 2005
- [BB91] BASS, Len J. ; BASS, Leonard J. ; COUTAZ, Joelle (Hrsg.): *Developing Software for the User Interface*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1991
- [BB03] BLECHSCHMITT, Eric ; BERNER, Uwe: A Dialog based User-Interface for Different Devices and Modalities. In: *International Workshop on Mobile Computing, IMC 2003*. Rostock, Germany, June 17-18 2003
- [BBC⁺04] BANAVAR, Guruduth ; BERGMAN, Lawrence ; CARDONE, Richard ; GAEREMYNCK, Vianney Chevalierand Y. ; GIRAUD, Frederique ; HALVERSON, Christine ; SHIN-ICHIHIROSE ; HORI, Masahiro ; KITAYAMA, Fumihiko ; KONDOH, Goh ; ASHISHKUNDU ; ONO, Kohichi ; SCHADE, Andreas ; SOROKER, Danny ; ; KIMWINZ: An Authoring Technology for Multidevice Web Applications. In: *IEEE Pervasive Computing* 3 (2004), July/september, Nr. 3, S. 83–93
- [BBGG89] BENNET, William ; BOIES, Stephen ; GOULD, John ; GREENE, Charles: Transformations on a dialog tree: Rule-based mapping of content to style. In: *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software andTechnology*. New York, USA : ACM, November 1989, S. 67–75
- [BBSS02] BERGMAN, Lawrence ; BANAVAR, Guruduth ; SOROKER, Danny ; SUSSMAN, Jeremy: Combining handcrafting and automatic generation of user-interfaces for pervasivedevices. In: KOLSKI, Christophe (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces III*. Dordrecht, The Netherlands : Kluwer Academic Publishers, May 2002, Kapitel 14, S. 155–166
- [BCCR04] BEDERSON, Ben B. ; CLAMAGE, Aaron ; CZERWINSKI, Mary P. ; ROBERTSON, George G.: DateLens: A Fisheye calendar interface for PDAs. In: *ACM Transactions on Computer-Human Interaction* 11 (2004), March, Nr. 1, S. 90–119
- [BCFM01] BONGIO, Aldo ; CERI, Stefano ; FRATERNALI, Piero ; MAURINO, Andrea: Modeling data entry and operations in WebML. In: *WebDB Bd. 1997*. Dallas, USA, 2001, S. 201
- [BCMP04] BERTI, Silvia ; CORREANI, Francesco ; MORI, Giulio ; PATERNO, Carmen: TERESA: a transformation-based environment for designing and developingmultidevice interfaces. In: *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM Press, 2004. – ISBN 1–58113–703–6, S. 793–794
- [BDRS97] BROWNE, Thomas ; DAVILA, David ; RUGABAR, Spencer ; STIREWALT, Kurt: Using declarative descriptions to model user interfaces with MASTERMIND. In: PATERNO, Fabio (Hrsg.) ; PALANQUE, Philippe (Hrsg.): *Formal methods in human computer interaction*. Berlin : Springer, 1997
- [Bec99] BECK, Kent: *Extreme programming explained*. Reading, Mass., USA : Addison-Wesley, 1999
- [Bed00] BEDERSON, Ben B.: Fisheye Menus. In: *Proceedings of UIST'2000*. New York, NY, USA : ACM Press, May 2000, S. 217–225
- [Bes92] BEST, John B.: *Cognitive Psychology*. 3rd. St. Paul, MN, USA : West Publishing Company, 1992
- [BGT05] BALFANZ, Dirk ; GRIMM, Matthias ; TAZARI, Saied: A Reference Architecture for Mobile Knowledge Management. In: *Dagstuhl Seminar 2005, Mobile Computing and Ambient Intelligence: The Challenge of Multimedia*. Wadern, Germany, May 2005

- [BH98] BEYER, Hugh ; HOLTZBLATT, Karen: *Contextual design: Defining customer-centered systems*. San Diego, USA : Academic Press, 1998
- [BH03] BARNES, Stuart J. ; HUFF, Sid L.: Rising sun: iMode and the wireless Internet. In: *Commun. ACM* 46 (2003), Nr. 11, S. 78–84. – ISSN 0001–0782
- [BHB04] BRINKMAN, Wilem-Paul ; HAAKMA, Reinder ; BOUWHUIS, Don G.: Consistency: a Factor that Links the Usability of Individual Interaction Components Together. In: *The Proceedings of Twelfth European Conference on Cognitive Ergonomics (ECCE-12)*, 2004, S. 57–64
- [BHL94] BODART, Francois ; HENNEBERT, Anne-Marie ; LEHEUREUX, Jean: A model-based approach to presentation: A continuum from task analysis to prototype. In: *Proceedings of DSV-IS'94*. Boca di Magra, IT, June 1994, S. 25–39
- [BHL99] BRAY, Tim ; HOLLANDER, Dave ; LAYMAN, Andrew: Namespaces in XML / World Wide Web Consortium (W3C). 1999 (<http://www.w3.org/TR/REC-xml-names>). – Recommendation
- [BHW92] BEIMEL, J. ; HÜTTNER, J. ; WANDKE, Hartmut: Kenntnisse von Programmierern auf dem Gebiet der Software-Ergonomie: Stand und Möglichkeiten zur Verbesserung. In: GEBERT, A. (Hrsg.): *Bericht über die 34. Fachtagung der Sektion Arbeits-, Betriebs- und Organisationspsychologie des BDP in Bad Lauterberg*. Stuttgart, Germany : Verlag für Angewandte Psychologie, 1992
- [BI03] BIEBER, Gerald ; IDE, Rüdiger: XyberScout: A platform for the efficient construction of mobile location aware information systems. In: SPRAGUE, Ralph H. (Hrsg.): *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*. Washington, USA : IEEE Computer Society Press, 2003, S. 297–305
- [Bir33] BIRKHOFF, George D.: *Aesthetic Measure*. Cambridge, MA, USA : Harvard University Press, 1933
- [BJR98] BOOCH, Grady (Hrsg.) ; JACOBSON, Ivar (Hrsg.) ; RUMBAUGH, James (Hrsg.): *UML konzentriert: die neue Standard-Objektmodellierungssprache anwenden*. 2. Auflage. Bonn : Addison-Wesley, 1998
- [BKKS04] BUTZ, Andreas (Hrsg.) ; KRAY, Christian (Hrsg.) ; KRÜGER, Antonio (Hrsg.) ; SCHMIDT, Albrecht (Hrsg.): *Workshop on Multi-User and Ubiquitous User Interfaces 2004 (MU3I)*. Funchal, Madeira, Januar 2004
- [BLC03] VAN DEN BERGH, Jan ; LUYTEN, Kris ; CONINX, Karin: A run-time system for context-aware multi-device user interfaces. In: *Proceedings of HCI International*, 2003, S. 308–312
- [BNFV02] BEIREKDAR, Abdo ; NOIRHOMME-FRAITURE, Monique ; VANDERDONCKT, Jean: KWARESMI - Knowledge-based Web Automated Evaluation with REconfigurable-guidelineS optiMization. In: *Proceedings of 9th International Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002*. Berlin, Germany : Springer-Verlag, June 2002
- [Boe86] BOEHM, Barry G.: A spiral model of software development and enhancement. In: *SIGSOFT Softw. Eng. Notes* 11 (1986), Nr. 4, S. 14–24. – ISSN 0163–5948
- [Bon68] BONSIÉPE, Gui A.: A method of quantifying order in typographic design. In: *Journal of Typographic Research* 2 (1968), S. 203–220
- [Boo92] BOOZ ALLAN & HAMILTON, INC.: NeXTStep vs other development environments / NeXT Computer, Inc. 1992. – Comparative Study
- [Bor50] BORING, Edwin G.: *History of experimental psychology*. New York, USA : Appleton-Century-Crofts, 1950
- [Bor89] BORTZ, Jürgen: *Statistik für Sozialwissenschaftler*. 3. Auflage. Berlin, Heidelberg : Springer Verlag, 1989
- [Bor01] BORCHERS, Jan: *A Pattern Approach to Interaction Design*. West Sussex, England : John Wiley & Sons Ltd, 2001 (Series in Software Design Patterns)

- [Bos04] BOSCH, Andi: *Java Server Faces*. München, Germany : Addison-Wesley, June 2004
- [Boy04] BOYERA, Stéphanie. *Personal communication at the W3C Multimodal Interaction Workshop at Sophia-Antipolis, France*. July 2004
- [Boy05] BOYER, John M.: XForms 1.1 / World Wide Web Consortium (W3C). 2005 (<http://www.w3.org/TR/xforms11/>). – W3C Working Draft
- [Bro96] BROOKE, John: SUS: A quick and dirty usability scale. In: JORDAN, P. (Hrsg.) ; B. THOMAS (Hrsg.) ; WEERDMEESTER, B. (Hrsg.) ; MCCLELLAND, I. (Hrsg.): *Usability evaluation in industry*. London, UK : Taylor and Francis, 1996, S. 189–194
- [BS03a] BLECHSCHMITT, Eric ; STRÖDECKE, Christoph: A Middleware For Conversational User Interfaces In Mobile Environments. In: *Proc. of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*. Darmstadt, Germany, March 2003
- [BS03b] BLECHSCHMITT, Eric ; STRÖDECKE, Christoph: Non Hierarchical Mergeable Dialogs. In: *10th International Conference on Human - Computer Interaction, HCI2003*. Crete, June 22-27 2003
- [BSF88] BARFIELD, Woodrow ; SANDFORD, James ; FOLEY, James D.: The mental rotation and perceived realism of computer-generated three-dimensional images. In: *Int. J. Man-Mach. Stud.* 29 (1988), Nr. 6, S. 669–684. – ISSN 0020–7373
- [BV02] BOUILLON, Laurent ; VANDERDONCKT, Jean: Retargeting web pages to other computing platforms with VAQUITA. In: VAN DEURSEN, Arie (Hrsg.) ; BURD, Elizabeth (Hrsg.): *9th Working Conference on Reverse Engineering (WCRE 2002)*. Richmond, VA, USA : IEEE Computer Society, October, 28 - November 1 2002, S. 339–345
- [BVE02] BOUILLON, Laurent ; VANDERDONCKT, Jean ; EISENSTEIN, Jacob: Model-based approaches to reengineering web pages. In: PRIBEANU, Costin (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Task Models and Diagrams for User Interface Design: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design- TAMODIA*. Bucharest, Romania : INFOREC Publishing House Bucharest, July, 18-19 2002, S. 86–95
- [BVNF02] BEIREKDAR, Abdo ; VANDERDONCKT, Jean ; NOIRHOMME-FRAITURE, Monique: A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In: KOLSKI, Christophe (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002*. Valenciennes, France : Kluwer Academics Pub., May, 15-17 2002, S. 337–348
- [BVS02] BOUILLON, Laurent ; VANDERDONCKT, Jean ; SOUCHON, Nathalie: Recovering alternative presentation models of a web page with VAQUITA. In: KOLSKI, Christophe (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces*. Valenciennes, France : Kluwer, May, 15-17 2002, S. 311–322
- [BWS⁺94] BYRNE, Michael D. ; WOOD, Scott D. ; SUKAVIRIYA, Piyawadee N. ; FOLEY, James D. ; KIERAS, David E.: Automating Interface Evaluation. In: *Human Factors in Computing Systems: Proceedings of CHI'94*. Reading, MA : Addison-Wesley, April 1994, S. 232–237
- [Car03a] CARD, Stuart: Information visualization. In: JACKO, Julie A. (Hrsg.) ; SEARS, Andrew (Hrsg.): *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*. Mahwah, NJ, USA : Lawrence Erlbaum Associates, 2003, Kapitel 28, S. 544–583
- [Car03b] CARROLL, John M.: *HCI models, theories and frameworks: Towards a multidisciplinary science*. San Francisco, USA : Morgan Kaufmann, 2003
- [CCB⁺03] CALVARY, Gaele ; COUTAZ, Joelle ; BOUILLON, Laurent ; FLORINS, Quentin ; MARUCCI, Luisa ; PATERNO, Fabio ; SANTORO, Carmen ; SOUCHON, Nathalie ;

- THEVENIN, David ; VANDERDONCKT, Jean: The CAMELEON reference framework / Cameleon Project: Plasticity of User Interfaces. 2003 (1.1). – Project Deliverable
- [CCT⁺02] CALVARY, Gaelle ; COUTAZ, Joelle ; THEVENIN, David ; LIMBOURG, Nathalie ; BOUILLON, Laurent ; FLORINS, Murielle ; VANDERDONCKT, Jean: Plasticity of User Interfaces: A Revised Reference Framework. In: PRIBEANU, Costin (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *First International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2002* Bd. Task Models and Diagrams for User Interface Design. Bucharest, Romania : INFORCE Publishing House, 2002, S. 127–134
- [CCT⁺03] CALVARY, Gaelle ; COUTAZ, Joelle ; THEVENIN, Daniel ; LIMBOURG, Laurent ; VANDERDONCKT, Jean: A unifying reference framework for multi-target user interfaces. In: *Interacting with Computers* 15 (2003), Nr. 3, S. 289–308
- [CD99] CLARK, James ; DEROSE, Steven: XML Path Language (XPath) / World Wide Web Consortium (W3C). 1999 (<http://www.w3.org/TR/xpath>). – Recommendation
- [CDJP04] CHOU, Wu ; DAHL, Deborah A. ; JOHNSTON, Michael ; PIERACCINI, Dave: EMMA: Extensible MultiModal Annotation markup language / World Wide Web Consortium (W3C). 2004. – W3C Working Draft
- [CDN88] CHIN, John P. ; DIEHL, Virginia A. ; NORMAN, Kent L.: Development of an instrument measuring user satisfaction of the human-computer interface. In: *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1988. – ISBN 0–201–14237–6, S. 213–218
- [CFB00] CERI, Stefano ; FRATERNALI, Piero ; BONGIO, Aldo: Web Modeling Language (WebML): a modeling language for designing Web sites. In: *Computer Networks*. Amsterdam, 2000
- [CGB00] CREASE, Murray ; GRAY, Philip D. ; BREWSTER, Stephen A.: A toolkit mechanism and context independent widgets. In: *Interactive Systems: Design, Specification, and Verification, 7th International Workshop DSV-IS*. Limerick, Ireland : Springer, June, 5-6 2000 (Lecture Notes in Computer Science 1946), S. 127–141
- [CGR82] CHI, M. T. H. ; GLASER, R. ; REES, E.: Expertise in problem solving. In: STERNBERG, R.J. (Hrsg.): *Advances in the psychology of human intelligence*. Hillsdale, N.J., USA : Erlbaum Associates, 1982
- [CHK⁺02] CLEMENS, Dirk ; HECK, Helmut ; KÜHN, Michael ; PERLICK, Olaf ; REINS, Frank: Individually Assisted Text Entry with Situational and Contextual Prediction. In: *Proceedings of the 8th International Conference ICCHP 2002*. Linz, Austria, 2002
- [CHNO86] CHENG, Patricia W. ; HOLYOAK, Keith J. ; NISBETT, Richard E. ; OLIVER, Lindsay M.: Pragmatic vs. syntactic approaches to training deductive reasoning. In: *Cognitive Psychology* 18 (1986), S. 293–328
- [CL96] CORAM, Todd ; LEE, Jim. *Experiences: A Pattern Language for User Interface Design*. <http://www.maplefish.com/todd/papers/Experiences.html>. (retrieved: 27.01.2006) 1996
- [CLC04a] CLERCKX, Tim ; LUYTEN, Kris ; CONINX, Karin: Generating context-sensitive multiple device interfaces from design. In: JACOB, Robert J. K. (Hrsg.) ; LIMBOURG, Quentin (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces IV, Proceedings of Fourth International Conference on Computer-Aided Design of User Interfaces*. Funchal, Madeira, Portugal : Kluwer, January 14-16 2004, S. 281–294
- [CLC04b] CLERCKX, Tim ; LUYTEN, Kris ; CONINX, Karin: The mapping problem back and forth: customizing dynamic models while preserving consistency. In: *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*. New York, NY, USA : ACM Press, 2004. – ISBN 1–59593–000–0, S. 33–42

- [CLV⁺03] CONINX, Karin ; LUYTEN, Kris ; VANDERVELPEN, Chris ; VEN DEN BERGH, Jan ; CREEMERS, Bert: Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In: CHITTARO, Luca (Hrsg.): *Proceedings of Fifth International Symposium on Human Computer Interaction with Mobile Devices (MobileHCI)*. Berlin, Germany : Springer-Verlag, September 2003, S. 256–270
- [CM94] COMBER, Tim ; MALTBY, John R.: Screen complexity and user design preference in windows applications. In: HOWARD, S. (Hrsg.) ; YOUNG, Y. K. (Hrsg.): *Proceedings of OZCHI'94*. Melbourne, Australia : CHISG, 1994, S. 133–137
- [CM96] COMBER, Tim ; MALTBY, John R.: Investigating Layout Complexity. In: *Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces, CADUI'96*. Namur, Belgium : Presses Universitaires de Namur, 1996
- [CMN83] CARD, Stuart K. ; MORAN, Thomas P. ; NEWELL, Alan: *The psychology of human computer interaction*. Hillsdale, New Jersey, USA : Lawrence Erlbaum Associates, 1983
- [CMP04] CORREANI, Francesco ; MORI, Giulio ; PATERNO, Fabio: Supporting flexible development of multi-device interfaces. In: BASTIDE, Rémi (Hrsg.) ; PALANQUE, Philippe A. (Hrsg.) ; ROTH, Jörg (Hrsg.): *Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS 2004* Bd. 3425. Berlin : Springer, July 2004, S. 346–362
- [Cov05] COVER, Robin. *XML Markup Languages for User Interface Definition*. <http://www.coverpages.com>. (retrieved: 27.01.2006) 2005
- [Cra43] CRAIK, Kenneth J. W.: *The nature of explanation*. Cambridge, UK : Cambridge University Press, 1943
- [CS66] CAMPBELL, Donald T. ; STANLEY, Julian C.: *Experimental and quasi-experimental designs for research*. Chicago, USA : Rand McNally, 1966
- [CS73] COOPER, L. A. ; SHEPARD, R. N.: Chronometric studies on the rotation of mental images. In: CHASE, W.G. (Hrsg.): *Visual information processing*. New York, USA : Academic Press, 1973
- [CSIK⁺82] CANFIELD SMITH, David ; IRBY, Charles ; KIRNBALL, Ralph ; VERPLANK, Bill ; HARSLEM, Eric: Designing the STAR User Interface. In: *Byte* 7 (1982), April, Nr. 4, S. 242–282
- [DA04] DEY, Anind K. ; ABOWD, Gregory D.: Support for the adapting applications and interfaces to context. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-aware Interfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, Kapitel 13, S. 262–296
- [DCCD03] DAASSI, Olfa ; CALVARY, Gaelle ; COUTAZ, Joelle ; DEMEURE, Alexandre: Comet: a new generation of widget for supporting user interface plasticity. In: *IHM 2003: Proceedings of the 15th French-speaking conference on human-computer interaction on 15eme Conference Francophone sur l'Interaction Homme-Machine*. New York, NY, USA : ACM Press, 2003. – ISBN 1–58113–803–2, S. 64–71
- [DFAB93] DIX, Alan ; FINLAY, Janet ; ABOWD, Gregory D. ; BEALE, Russell: *Human-computer interaction*. Herfordshire, UK : Prentice Hall, 1993
- [DFAM02] DEARDEN, Andy ; FINLAY, Janet ; ALLGAR, Elizabeth ; MCMANUS, Barbara: Using Pattern Languages in Participatory Design. In: T.BINDER (Hrsg.) ; J.GREGORY (Hrsg.) ; I.WAGNER (Hrsg.): *Proceedings of Participatory Design Conference (PDC02)*. Malmö, Sweden, June, 23–25 2002
- [DFB92] DIX, Alan ; FINLAY, Janet ; BEALE, Russel: Analysis of user behaviour as time series. In: MONK, A. (Hrsg.) ; DIAPER, D. (Hrsg.) ; HARRISON, M. (Hrsg.): *Proceedings of HCI'92: People and Computers VII*. Cambridge, USA : Cambridge University Press, September 1992, S. 249–444
- [DHM98] DRÖSCHEL, Wolfgang ; HEUSER, Walter ; MIDDERHOFF, Rainer: *Inkrementelle und objektorientierte Vorgehensweisen mit dem V-Modell 97*. München : Oldenbourg, 1998

-
- [Dix91] DIX, Alan: *Formal methods for interactive systems*. London, UK : Academic Press Ltd, 1991 (Computer and People Series)
- [DK04] DENIS, Charles ; KARSENTY, Laurent: Inter-Usability of multi-device systems — A conceptual framework. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform applications and context-aware interfaces*. Chichester, West Sussex, England : Wiley & Sons, Ltd., 2004, Kapitel 17, S. 374–385
- [DKMR03] DUBINKO, Micah ; KLOTZ, Leigh L. ; MERRICK, Roland ; RAMAN, T. V.: XForms 1.0 — W3C Recommendation / World Wide Web Consortium (W3C). 2003. – Recommendation
- [Doy90] DOYLE, J. R.: Naive users and the Lotus interface: A field study. In: *Behaviour & Information Technology* 9 (1990), Nr. 1, S. 81–89
- [DS05] DE SOUZA, Clarisse S.: *The semiotic engineering of human-computer interaction*. Cambridge, Mass, USA : MIT Press, 2005 (Acting with technology)
- [Dub03] DUBINKO, Micah: *XForms Essentials*. 1st Edition. Sebastopol, CA, USA : O'Reilley, 2003
- [Dun45] DUNCKER, K.: On problem-solving. In: *Psychological Monographs* 58 (1945), Nr. 270, S. whole
- [Eas84] EASON, K. D.: Towards the experimental study of usability. In: *Behavior and Information Technology* 3 (1984), Nr. 2, S. 133–143
- [Eck06] ECKERT, Jason: *Microsoft Windows Vista Guide*. Boston, MA, USA : Course Technology, 2006
- [ECM99] ECMA-262: ECMAScript Language Specification / ECMA International. Geneva, Switzerland, 1999 (262). – International Standard
- [EK93] EYSENCK, Michael W. ; KEANE, Mark T.: *Cognitive Psychology*. 4th. East Sussex, UK : Lawrence Earlbaum Ass. Ltd., 1993
- [EM87] EBERTS, R.E. ; MACMILLAN, A.G.: Longitudinal study of a distributed system. In: *Proceedings of Human Factors Society 28th Annual Meeting*, 1987, S. 704–708
- [EM02] ENGE, Marita ; MASSOW, Sascha: Needs for assistance of visually and physically disabled and non-disabled persons when using money-/cash dispensers. In: DE WAARD, D. (Hrsg.) ; WELKERT, C. (Hrsg.) ; HOONHOUT, J. (Hrsg.) ; REMERKERS, J. (Hrsg.): *Human-Computer Interaction: Education, Research and Application in the 21st Century*. Maastricht, The Netherlands : Shaker Verlage, 2002, S. 263–266
- [EMB03] EMBASSI. *Elektronische Multimodale Bedienassistentz*. <http://www.embassi.de>. (retrieved: 27.01.2006) 2003
- [Enc80] ENCARNAÇÃO, José L.: *Die Entstehung und Entwicklung des Graphischen Kernsystems GKS Geräteunabhängige Systeme: Computer Graphics und Portabilität oder Das Graphische Kernsystem GKS*. München, Germany : Oldenbourg Verlag, 1980 (Buchreihe „Darmstädter Kolloquium“)
- [Enc01] ENCARNAÇÃO, José L.: The next generation in computer supported interaction and communication. In: OBERQUELLE, Horst (Hrsg.) ; OPPERMANN, Reinhard (Hrsg.) ; KRAUSE, J. (Hrsg.): *Mensch & Computer*. Stuttgart, Germany : Teubner, March 2001, S. 23–24
- [Enc03] ENCARNAÇÃO, José L.: Die Integration des Menschen in intelligente IT-Umgebungen. In: WARNECKE, Hans J. (Hrsg.) ; BULLINGER, Hans-Jörg (Hrsg.): *Kunststück Innovation*. Berlin, Germany : Springer, 2003, S. 36–45
- [Eng01] VAN ENGERS, Tom M.: *Knowledge management: The role of mental models in business systems design*. Amsterdam, The Netherlands, Department of Computer Science, Vrije Universiteit, Diss., 2001
-

- [ER86] EMBREY, David E. ; REASON, James: The application of cognitive models to the evaluation and prediction of human reliability. In: *Proceedings of the international topical meeting on advances in human factors in nuclear power systems*. Knoxville, TN, USA : American Nuclear Society, April 1986, S. 292–301
- [ES95] ELWERT, Thomas ; SCHLUNGBAUM, Egbert: Modelling and generation of graphical user interfaces in the TADEUS approach. In: *Design, Specification and Verification of Interactive Systems (DSVIS)*. Vienna, AT, 1995, S. 193–208
- [EVP00] EISENSTEIN, Jacob ; VANDERDONCKT, Jean ; PUERTA, Angel: Adapting to mobile contexts with user-interface modeling. In: *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)*. Monterey, California, USA : IEEE Computer Society, December, 7-8 2000, S. 83ff
- [EVP01] EISENSTEIN, Jacob ; VANDERDONCKT, Jean ; PUERTA, Angel: Applying model-based techniques to the development of UIs for mobile computers. In: *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*. New York, NY, USA : ACM Press, 2001. – ISBN 1-58113-325-1, S. 69–76
- [FAA01] FAROOQ ALI, Mir ; ABRAMS, Marc: Simplifying construction of multi-platform user interfaces using UIML. In: *Proceedings of the UIML'2001*. Paris, France : Aristote, 2001
- [FAPQA04] FAROOQ ALI, Mir ; PÉREZ-QUINONES, Manuel A. ; ABRAMS, Marc: Building multi-platform user interfaces with UIML. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-aware Interfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 11–25
- [FB87] FLECCHIA, Mark A. ; BERGERON, R. D.: Specifying complex dialogs in ALGAE. In: *CHI '87: Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*. New York, NY, USA : ACM Press, 1987. – ISBN 0-89791-213-6, S. 229–234
- [FCKKM91] FOLEY, James D. ; CHUL KIM, Won ; KOVACEVIC, Srdjan ; MURRAY, Kevin: GUIDE — An Intelligent User Interface Design Environment. In: SULLIVAN, Joseph W. (Hrsg.) ; TYLER, Sherman W. (Hrsg.): *Architectures for Intelligent Interfaces: Elements and Prototypes*. New York, NY, USA : Addison-Wesley, 1991, S. 339–384
- [FD82] FOLEY, James D. ; VAN DAM, Andries: *Fundamentals of interactive computer graphics*. Reading, Massachusetts : Addison-Wesley, 1982 (System Programming Series)
- [FDFH90] FOLEY, James D. ; VAN DAM, Andries ; FEINER, Steven K. ; HUGHES, John F.: *Computer Graphics: Principles and Practice*. 2nd Edition. Reading, MA, USA : Addison-Wesley, 1990
- [FGK88] FOLEY, James D. ; GIBBS, C. ; KOVACEVIC, S.: A knowledge-based user interface management system. In: *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1988. – ISBN 0-201-14237-6, S. 67–72
- [FH02] FORKL, Yves ; HELLENSCHMIDT, Michael: Mastering Agent Communication in EMBASSI on the Basis of a Formal Ontology. In: *Proceedings of the ISCA Tutorial and Research Workshop, Multimodal Dialogue in Mobile Environments*. Kloster Irsee, Germany, June 2002
- [Fin00] FINCHER, Sally. *HCI Pattern-Form Gallery*. <http://www.cs.kent.ac.uk/people/staff-/safpatterns/gallery.html>. retrieved: 27.01.2006 2000
- [Fis22] FISHER, R. A.: The goodness of fit of regression formulae, and the distribution of regression coefficients. In: *Journal of the Royal Statistical Society* 85 (1922), S. 597–612
- [Fit54] FITTS, Paul M.: The information capacity of the human motor system in controlling the amplitude of movement. In: *Journal of Experimental Psychology* 47 (1954), S. 381–391

-
- [Fol04] FOLEY, James D.: *How good is good enough?* Presentation. October 2004. – Darmstadt, Germany
- [FP64] FITTS, Paul M. ; PETERSON, J. R.: Information capacity of discrete motor responses. In: *Journal of Experimental Psychology* 67 (1964), S. 103–112
- [FP67] FITTS, Paul M. ; POSNER, Michael I.: *Human Performance*. Belmont, CA, USA : Brooks Cole, 1967
- [FP99] FARENC, Christelle ; PALANQUE, Philippe A.: A Generic Framework based on Ergonomics Rules for Computer Aided Design of User Interface. In: VANDERDONCKT, Jean (Hrsg.) ; PUERTA, Angel R. (Hrsg.): *Computer-Aided Design of User Interfaces II, Proceedings of the Third International Conference of Computer-Aided Design of User Interfaces*. Louvain-la-Neuve, Belgium : Kluwer, October, 21-23 1999, S. 281–292
- [FP00] FRATERNALI, Piero ; PAOLINI, Paolo: Model-driven development of Web applications: the AutoWeb system. In: *ACM Trans. Inf. Syst.* 18 (2000), Nr. 4, S. 323–382. – ISSN 1046–8188
- [FR82] FELDMAN, Michael B. ; ROGERS, George T.: Toward the design and development of style-independent interactive systems. In: *Proceedings of the 1982 conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1982, S. 111–116
- [Fro72] FROST, N.: Encoding and retrieval in visual memory tasks. In: *Journal of Experimental Psychology* 95 (1972), S. 317–326
- [FTV04] FLORINS, Murielle ; TREVISAN, Daniela G. ; VANDERDONCKT, Jean: The continuity property in mixed reality and multiplatform systems: A comparative study. In: *Proceedings of the 4th Int. Workshop on Computer-Aided Design of User Interfaces-CADUI'04*. Funchal, Madeira, Portugal, January 13-16 2004, S. 323–333
- [FW04] FALLSIDE, David C. ; WALMSLEY, Priscilla: XML Schema Part 0: Primer Second Edition / World Wide Web Consortium (W3C). 2004 (<http://www.w3.org/TR/xmlschema-0/>). – W3C Recommendation
- [Gag66] GAGNÉ, Robert M.: *The conditions of learning*. New York, USA : Holt, Rinehart, and Winston, 1966
- [GBL03] GEAREMYNCK, Yves ; BERGMAN, Lawrence D. ; LAU, Tessa: MORE for less: Model recovery from visual interfaces for multi-device application design. In: *Proceedings of Conference on Intelligent User Interfaces (IUI'03)*. New York, USA : ACM Press, January 2003, S. 69–76
- [GBM⁺99] GRIFFITHS, Tony ; BARCLAY, Peter J. ; MCKIRDY, Jo ; PATON, Norman W. ; GRAY, Philip D. ; KENNEDY, Jessie B. ; COOPER, Richard ; GOBLE, Carole A. ; WEST, Adrian ; SMYTH, Michael: Teallach: a model-based user interface development environment for object databases. In: PATON, Norman W. (Hrsg.) ; GRIFFITHS, Tony (Hrsg.): *User Interfaces to Data Intensive Systems, UIDIS 1999*. Edinburgh, UK : IEEE Computer Society, September, 5-5 1999, S. 86–96
- [GC87] GARDINER, Margret M. ; CHRISTIE, Bruce: *Applying Cognitive Psychology to User-Interface Design*. Chichester, UK : Wiley & Sons, 1987 (Wileys Series in Information Processing)
- [GH80] GICK, Mary L. ; HOLYOAK, Keith J.: Analogical problem solving. In: *Cognitive Psychology* 12 (1980), S. 306–355
- [GH83] GICK, Mary L. ; HOLYOAK, Keith J.: Schema induction and analogical transfer. In: *Cognitive Psychology* 15 (1983), S. 1–38
- [GH00] GRUNDY, John ; HOSKING, John: Developing Adaptable User Interfaces for Component-Based Systems. In: *First Australasian User Interface Conference*. Canberra, Australia, 2000, S. 17
-

- [GH04] GILROY, Stephen W. ; HARRISON, Michael D.: Using interaction style to Match the ubiquitous user interface to the device-to-hand. In: *Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI-DSVIS'04)* IFIP, 2004
- [GHJV94] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass, USA : Addison-Wesley, 1994
- [GL83] GOULD, John D. ; LEWIS, Clayton: Designing for usability key principles and what designers think. In: *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press, 1983. – ISBN 0-89791-121-0, S. 50–53
- [GL05] VAN DER GEEST, Thea ; LOORBACK, Nicole: Testing the Visual Consistency of Web Sites. In: *Technical Communication* 52 (2005), Nr. 1, S. 27–36
- [GLS05] GANTENBEIN, C. ; LEBER, M. ; S.PORTMANN. *JAXFront Konzepte*. <http://www.jaxfront.com>. (retrieved: 21.01.2006) 2005
- [GMR04] GUNDELSWEILER, Fedrik ; MEMMEL, Thomas ; REITERER, Harald: Agile Usability Engineering. In: KEIL-SLAWIK, R. (Hrsg.) ; SELKE, H. (Hrsg.) ; SZWILLUS, G. (Hrsg.): *Proceedings of Mensch & Computer 2004: Allgegenwärtige Interaktion*. München, Germany : Oldenbourg Verlag, September 2004
- [Gol89] GOLDSTEIN, Bruce: *Sensation and Perception*. Third. Belmont, CA, USA : Wadsworth Publishing, 1989
- [Göt95] GÖTZE, Rainer: *Teubner Texte zur Informatik*. Bd. 14: *Dialogmodellierung für multimediale Benutzerschnittstellen*. Stuttgart : Teubner Verlagsgesellschaft, 1995
- [Gra00] GRAY, Wayne D.: The Nature and Processing of Errors in Interactive Behavior. In: *Cognitive Science* 24 (2000), Nr. 2, S. 205–248
- [Gre85] GREEN, Mark: Report on Dialogue Specification Tools. In: PFAFF, Günther E. (Hrsg.): *User Interface Management Systems*. Berlin, Germany : Springer, 1985, S. 9–20
- [Gru89] GRUDIN, Jonathan: The case against user interface consistency. In: *Communications of the ACM* 32 (1989), Nr. 10, S. 1164–1173
- [Gru92] GRUDIN, Jonathan: Consistency, standards, and formal approaches to interface development and evaluation: A note on Wiecha, Bennett, Boies, Gould, and Greene. In: *ACM Transactions on Information Systems* 10 (1992), January, Nr. 1, S. 103–111
- [GS83] GENTNER, Derdre A. ; STEVENS, Albert L.: *Mental Models*. Hillsdale, NJ, USA : Lawrence Erlbaum Associates, 1983
- [GSK93] GAY, Wayne D. ; SABNANI, Haresh ; KIRSCHENBAUM, Susan S.: Review of the book “Human Error”. In: *International Journal of Man-Machine Studies* 39 (1993), S. 1056–1057
- [GTB02] GRIMM, Matthias ; TAZARI, Saied ; BALFANZ, Dirk: Towards a Framework for mobile knowledge management. In: *Proceedings of the 4th international conference on Practical Aspects of Knowledge Management*. Vienna, Austria, December 2002
- [GVRV02] GROLAUX, Donatien ; VAN ROY, Peter ; VANDERDONCKT, Jean: FlexClock, a Plastic Clock Written in Oz with the QtK toolkit. In: *TAMODIA '02: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design*, INFOREC Publishing House Bucharest, 2002. – ISBN 973-8360-01-3, S. 135–142
- [GW04] GAJOS, Krzysztof ; WELD, Daniel S.: SUPPLE: automatically generating user interfaces. In: VANDERDONCKT, Jean (Hrsg.) ; NUNES, Nuno J. (Hrsg.) ; RICH, Charles (Hrsg.): *Proceedings of the 2004 International Conference on Intelligent User Interfaces*. Funchal, Madeira, Portugal : ACM, January, 13-16 2004, S. 93–100

- [GWW05] GAJOS, Krzysztof ; WU, Anthony ; WELD, Daniel S.: Cross-Device Consistency in Automatically Generated User Interfaces. In: *Proceedings of 2nd Workshop on Multi-User and Ubiquitous User Interfaces(MU3I)*. San Diego, CA, USA, January 9 2005
- [GZ04] GRUNDY, John ; ZOU, Wenjing: AUIT: Adaptable user interface technology, with extended Java ServerPages. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-awareInterfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 149–167
- [Haa93] DE HAAN, Geert: Formal representation of human-computer interaction. In: VAN DER VEER, G. C. (Hrsg.) ; WHITE, T. N. (Hrsg.) ; ARNOLD, A. G. (Hrsg.): *Proceedings of Human-Computer Interaction: Preparing for the nineties*. Amsterdam, The Netherlands : SIC, 1993, S. 95–112
- [Haa00] DE HAAN, Geert: *ETAG, A formal model of competence knowledge for user-interface design*, Vrije Universiteit Amsterdam, Diss., Oktober 2000
- [Her05] HEROLD, Frank: *Entwurf und Realisierung einer API für plattformunabhängige UserInterfaces*. Wiesbaden, Germany, Fachhochschule Wiesbaden, durchgeführt an der TU Darmstadt, Diplomarbeit, Februar 2005
- [HH89] HARTSON, H. R. ; HIX, Deborah: Human-computer interface development: concepts and systems for its management. In: *ACM Comput. Surv.* 21 (1989), Nr. 1, S. 5–92. – ISSN 0360–0300
- [HH93] HIX, Deborah ; HARTSON, H. R.: *Developing user interfaces: ensuring usability through product & process*. John Wiley & Sons, Inc., 1993
- [Hic52] HICK, W. E.: On the rate of gain of information. In: *Quarterly Journal of Experimental Psychology* 4 (1952), S. 11–36
- [HK01] HERFET, Thorsten ; KIRSTE, Thomas: EMBASSI – Multimodal Assistance for Infotainment & Service Infrastructures. In: *Proceedings der Statustagung der Leitprojekte „Mensch-Technik-Interaktion“*. Saarbruecken, Germany, October 26-27 2001, S. 35–44
- [HKR03] HELLENSCHMIDT, Michael ; KIRSTE, Thomas ; RIEGER, Thomas: An agent-based Approach to Distributed User Profile Management within aMulti-Modal Framework. In: *Proceedings of the IMC2003*. Rostock, Germany, 2003
- [HNM94] HOLYOAK, Keith J. ; NOVICK, Laura R. ; MELZ, Eric R.: Component processes in analogical transfer: Mapping, pattern completionand adaptation. In: HOLYOAK, K. J. (Hrsg.) ; BARNDEN, J. A. (Hrsg.): *Advances in Connectionist and Neural Computation Theory, Vol.2: AnalogicalConnections*. Norwood, NJ, USA : Ablex, 1994, S. 113–180
- [Hol04] HOLZNER, Steve: *Eclipse*. Sebastopol, USA : O'Reilly, 2004
- [HR98] HACKOS, Jo-Ann T. ; REDISH, Janice C.: *User and task analysis for interface design*. New York, NY, USA : John Wiley & Sons, Inc., 1998
- [HS88] HART, Sandra G. ; STAVELAND, Lowell E.: Development of a NASA-TLX (Task Load Index): Results of Empirical andTheoretical Research. In: HANCOCK, P. A. (Hrsg.) ; MESHKATI, N. (Hrsg.): *Human Mental Workload*. Amsterdam, The Netherlands : Elsevier, 1988, S. 139–183
- [HSL85] HAYES, Philip J. ; SZEKELY, Pedro A. ; LERNER, Richard A.: Design alternatives for user interface management sytems based on experiencewith COUSIN. In: *CHI '85: Proceedings of the SIGCHI conference on Human factors in computingsystems*. New York, NY, USA : ACM Press, 1985. – ISBN 0–89791–149–0, S. 169–175
- [Hüb90] HÜBNER, Wolfgang: *Entwurf graphischer Benutzerschnittstellen: ein objektorientiertes Interaktionsmodellzur Spezifikation graphischer Dialoge.*, Technische Universität Darmstadt, Diss., 1990

- [Hud96] HUDLICKA, Eva: Requirements elicitation with indirect knowledge elicitation techniques: comparison of three models. In: *Proceedings of the second international conference on requirement engineering (ICRE'96)*, IEEE Computer Society Press, 1996, S. 4–11
- [HV04] HONKALA, Mikko ; VUORIMAA, Petri: A Configurable XForms Implementation. In: *Proceedings of IEEE Multimedia Software Engineering*. Miami, Florida, USA, December 13-15 2004
- [HVV91] DE HAAN, Geert ; VAN DER VEER, Gerrit C. ; VAN VLIET, Jan C.: Formal modelling techniques in human-computer interaction. In: *Acta Psychologica* 78 (1991), S. 26–76
- [Hym53] HYMAN, R.: Stimulus information as a determinant of reaction time. In: *Journal of Experimental Psychology* 45 (1953), S. 188–196
- [IH01] IVORY, Melody Y. ; HEARST, Marti A.: The state of the art in automating usability evaluation of user interfaces. In: *ACM Comput. Surv.* 33 (2001), Nr. 4, S. 470–516. – ISSN 0360–0300
- [Int05a] INTERNATIONAL COMMITTEE FOR INFORMATION TECHNOLOGY STANDARDS. *ANSI/INCITS 389: American National Standard - Information technology - Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Universal Remote Console*. Standard. September 2005
- [Int05b] INTERNATIONAL COMMITTEE FOR INFORMATION TECHNOLOGY STANDARDS. *ANSI/INCITS 390: American National Standard - Information technology - Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: User Interface Socket Description*. Standard. September 2005
- [Int05c] INTERNATIONAL COMMITTEE FOR INFORMATION TECHNOLOGY STANDARDS. *ANSI/INCITS 391: American National Standard - Information technology - Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Presentation Templates*. Standard. September 2005
- [Int05d] INTERNATIONAL COMMITTEE FOR INFORMATION TECHNOLOGY STANDARDS. *ANSI/INCITS 392: American National Standard - Information technology - Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Target Properties Sheet*. Standard. September 2005
- [Int05e] INTERNATIONAL COMMITTEE FOR INFORMATION TECHNOLOGY STANDARDS. *ANSI/INCITS 393: American National Standard - Information technology - Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Resource Description*. Standard. September 2005
- [ISO94] ISO/IEC 7942: ISO/IEC 7942: Information technology – Computer graphics and image processing– Graphical Kernel System (GKS) – Part 1: Functional description / International Organization for Standardization. 1994 (ISO/IEC 9742-1:1994). – International Standard
- [ISO99a] ISO 13407: User centered design process for interactive systems / International Organization for Standardization. Geneva, Switzerland, 1999 (13407). – International Standard
- [ISO99b] ISO 14598: Information Technology — Software Product Evaluation / International Standards Organization. Geneva, Switzerland, 1999 (14598). – International Standard
- [ISO99c] ISO 18529: Human-centered lifecycle process for interactive systems / International Standards Organization. Geneva, Switzerland, 1999 (18529). – International Standard

-
- [ISO99d] ISO/IEC 9126: Quality Characteristics and Guidelines for the User / International Standards Organization. Geneva, Switzerland, 1999 (9126). – International Standard
- [ISO01] ISO 9241: Ergonomic requirements for office work with visual display terminals (VDTs) / International Standards Organization. Geneva, Switzerland, 2001 (9241). – International Standard
- [IST02] IST - INFORMATION SOCIETY: Information Society Technologies - A thematic priority for Research and Development under the Specific Programme “Integrating and strengthening the European Research Area” in the Community sixth Framework Programme- 2003-2004 Workprogramme / The European Commission. 2002. – Forschungsbericht
- [IST03] IST ADVISORY GROUP: Ambient Intelligence: from vision to reality / The European Commission. 2003. – Forschungsbericht
- [Jac92] JACOBSON, Ivar: *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading, Mass., USA : Addison-Wesley, 1992
- [Jak60] JAKOBSON, Roman: Linguistics and Poetics. In: SEBEOK, T.A. (Hrsg.): *Style in Language*. Cambridge, MA : MIT Press, 1960, S. 350–377
- [JL83] JOHNSON-LAIRD, Philip N.: *Cognitive science series*. Bd. 6: *Mental Models*. Cambridge, Massachusetts : Harvard University Press, 1983
- [Joh00] JOHNSON, Jeff: *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Oxford, UK : Morgan Kaufman, 2000 (The Morgan Kaufmann Series in Interactive Technologies)
- [JRV⁺er] JOHNSON, Jeff ; ROBERTS, Teresa L. ; VERPLANK, William ; SMITH, David C. ; IRBY, Charles ; BEARD, Marian ; MACKEY, Kevin: The Xerox Star: A Retrospective. In: *IEEE Computer* 22 (IEEE Computer), Nr. 9, S. 11–29
- [JSES04] JAVAHERY, Homa ; SEFFAH, Ahmed ; ENGELBERG, Daniel ; SINNIG, Daniel: Migrating User Interfaces Across Platforms Using HCI Patterns. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-aware Interfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 241–259
- [Kas82] KASIK, David J.: A user interface management system. In: *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1982. – ISBN 0–89791–076–1, S. 99–106
- [Kay02] KAY, Alan: Der Schulspaßmacher. In: *DIE ZEIT* 47 (2002), S. n.a.
- [KB02] KOTHARI, Ravi ; BASAK, Jayanta: Perceptually motivated measures for capturing proximity of web page elements: Towards automated evaluation of web page layouts. In: *Proceedings of World Wide Web Conference (WWW2002)*. Honolulu, Hawaii, USA, 2002
- [KBR78] KOSSLYN, Stephen M. ; BALL, Thomas M. ; REISER, Brian J.: Visual images preserve metric spatial information: Evidence from studies of image scanning. In: *Journal of Experimental Psychology: Human Perception and Performance* 4 (1978), S. 47–60
- [KD95] KARAT, John ; DAYTON, Tom: Practical education for improving software usability. In: *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1995. – ISBN 0–201–84705–1, S. 162–169
- [Kel87] KELLOGG, Wendy A.: Conceptual consistency in the user interface: Effects on user performance. In: *Proceedings of International Conference on Human-Computer Interaction (INTERACT'87)*. Stuttgart, Germany, September 1–4 1987
- [Kel89] KELLOGG, Wendy A.: The Dimensions of Consistency. In: NIELSEN, Jakob (Hrsg.): *Coordinating User Interfaces for Consistency*. Academic Press, 1989, S. 9–20
- [KF88] KASS, Robert ; FININ, Timothy W.: A general user modelling facility. In: *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1988. – ISBN 0–201–14237–6, S. 145–150
-

- [KF90] KIM, Won C. ; FOLEY, James D.: DON: user interface presentation design assistant. In: *UIST '90: Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interfacesoftware and technology*. New York, NY, USA : ACM Press, 1990. – ISBN 0-89791-410-4, S. 10–20
- [KF93] KIM, Won C. ; FOLEY, James D.: Providing high-level control and expert assistance in the user interfacepresentation design. In: *CHI '93: Proceedings of the SIGCHI conference on Human factors in computingsystems*. New York, NY, USA : ACM Press, 1993. – ISBN 0-89791-575-5, S. 430–437
- [Koc05] KOCH, Christine: *Microsoft Windows “Longhorn” — Die Neuerungen im Überblick*. Unterschleißheim, Germany : Microsoft Press Deutschland, 2005
- [Kop04] KOPP, Timo: *Entwicklung eines Systems zur plattformunabhängigen Spezifikation und Darstellung von User Interfaces*. Friedberg, Germany, Fachhochschule Friedberg, durchgeführt an der TU Darmstadt, Diplomarbeit, August 2004
- [Kos80] KOSSLYN, Stephen M.: *Image and mind*. Cambridge, Mass., USA : Harvard University Press, 1980
- [Kov93] KOVACEVIC, Srdjan: TACTICS for user interface Design: Coupling the compositional and the transformational approach / US West Advanced Technologies. 1993 (At-0910-002763-01.00). – Technical Report
- [Kov94] KOVACEVIC, Srdjan: TACTICS – A model-based framework for multimodal interaction. In: *Proc. of the AAAI Spring Symposium on Intelligent Multi-Media Multi-Modal Systems*, 1994
- [KP85] KIERAS, David ; POLSON, Peter G.: An approach to the formal analysis of user complexity. In: *International Journal of Man-Machine Studies* 22 (1985), Nr. 4, S. 365–394
- [KP02] KEROENEN, Heikki ; PLOMP, Johan: Adaptive runtime layout of hierarchical UI components. In: *NordiCHI '02: Proceedings of the second Nordic conference on Human-computerinteraction*. New York, NY, USA : ACM Press, 2002. – ISBN 1-58113-616-1, S. 251–254
- [Kru04a] KRUCHTEN, Philippe: *The Rational Unified Process*. 3rd. Reading, MA, USA : Addison-Wesley, 2004
- [Kru04b] KRUEGER, Guido: *Handbuch der Java-Programmierung*. München, Germany : Addison-Wesley, Dezember 2004
- [KRW⁺04] KLYNE, Graham ; REYNOLDS, Franklin ; WOODROW, Chris ; AND JOHAN HJELM, Hidetaka O. ; BUTLER, Mark H. ; TRAN, Luu: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0 / World Wide Web Consortium (W3C). 2004. – W3C Recommendation
- [Küs03] KÜSTER, Mark W.: Internationalisierung und Lokalisierung / Saphor GmbH. Tübingen, Germany, 2003. – Technical Report
- [Lan05] LANZ, Simone: *Modellierung von Interaktionen in einer remote Benutzerschnittstelle*. Mannheim, Germany, Fachhochschule Mannheim, Fachbereich Informationstechnik, Studiengang Technische Informatik, Diplomarbeit, December 2005
- [LB99a] LIE, Hakon W. ; BOS, Bert: *Cascading Style Sheets: Designing for the Web*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 0201596253
- [LB99b] LIE, Hakon W. ; BOS, Bert: Cascading Style Sheets, level 1 / World Wide Web Consortium (W3C). 1999. – W3C Recommendation 17 Dec 1996, revised 11 Jan 1999
- [Le 53] LE CORBUSIER: *Der Modulor: Darstellung eines in Architektur und Technik allgemein anwendbaren harmonischen Maßes im menschlichen Maßstab*. Stuttgart, Germany : J. G. Cotta'sche Buchhandl. Nachf, 1953
- [Len01] LENTH, Russel V.: Some practical guidelines for effective sample-size determination. In: *The American Statistician* 55 (2001), S. 187–193

- [LF97] LABROU, Yannis ; FININ, Timothy W.: A Proposal for a new KQML Specification / Computer Science and Electrical Engineering Department, UMBC. Baltimore, MD, USA, 1997 (TR CS-97-03). – Forschungsbericht
- [LM04] LEWIS, Rhys ; MERRICK, Roland: Content Selection for Device Independence (DI-Select) 1.0 / World Wide Web Consortium (W3C). 2004. – W3C Working Draft
- [LNR87] LAIRD, John E. ; NEWELL, Allen ; ROSENBLOOM, Paul: SOAR: an architecture for general intelligence. In: *Artificial Intelligence* 33 (1987), S. 1–64
- [LS96] LONCZEWSKI, Frank ; SCHREIBER, Siegfried: The FUSE-System : an Integrated User Interface Design Environment. In: VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium : Presses Universitaires de Namur, June, 5-7 1996 (Travaux de l'Institut d'Informatique), S. 37–56
- [LSN93] LUO, Ping ; SZEKELY, Pedro A. ; NECHES, Robert: Management of interface design in HUMANOID. In: *Proceedings of InterCHI'93*, 1993, S. 107–114
- [LV04] LIMBOURG, Quentin ; VANDERDONCKT, Jean: Addressing the Mapping Problem in User Interface Design with UsiXML. In: SLAVÍK, Pavel (Hrsg.) ; PALANQUE, Philippe A. (Hrsg.): *Task Models and Diagrams for User Interface Design: Proceedings of the Third International Workshop on Task Models and Diagrams for User Interface Design- TAMODIA 2004*. Prague, Czech Republic : ACM Press, November, 15-16 2004, S. 155–163
- [LVC02] LUYTEN, Kris ; VANDERVELPEN, Chris ; CONINX, Karin: Migratable User Interface Descriptions in Component Based Development. In: FORBRIG, Peter (Hrsg.) ; LIMBOURG, Quentin (Hrsg.) ; URBAN, Bodo (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Interactive Systems. Design, Specification, and Verification, 9th International Workshop, DSV-IS 2002*. Rostock Germany : Springer, June, 12-14 2002 (Lecture Notes in Computer Science), S. 44–58
- [LVM⁺04] LIMBOURG, Quentin ; VANDERDONCKT, Jean ; MICHOTTE, Benjamin ; BOUILLON, Laurent ; LOPÉZ-JAQUERO, Victor: UsiXML: a language supporting multipath development of user interfaces. In: BASTIDE, Rémi (Hrsg.) ; PALANQUE, Philippe A. (Hrsg.) ; ROTH, Jörg (Hrsg.): *Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS 2004* Bd. 3425. Hamburg, Germany : Springer, July, 11-13 2004, S. 220–220
- [Maa83] MAASS, Susanne: Why systems transparency? In: GREEN, Thomas R. G. (Hrsg.) ; PAYNE, Stephen J. (Hrsg.) ; VAN DER VEER, Gerrit C. (Hrsg.): *The Psychology of Computer Use*. Orlando, FL, USA : Academic Press, Inc., 1983, S. 19–28
- [Mac89] MACKENZIE, I. S.: A note on the information-theoretic basis for Fitts' law. In: *Journal of Motor Behavior* 21 (1989), S. 323–330
- [Mah96] MAHAJAN, Rohit: *A family of user interface consistency checking tools*. College Park, MD, USA, University of Maryland, Master of Science, 1996
- [Mar01] MARSH, Jonathan: XML Base / World Wide Web Consortium (W3C). 2001. – W3C Recommendation 27 June 2001
- [May99] MAYHEW, Deborah J.: *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design*. San Francisco : Morgan Kaufmann Publishers, 1999 (The Morgan Kaufmann Series in Interactive Technologies)
- [MF02] MASSINK, Mieke ; FACONTI, Giorgio P.: A reference framework for continuous interaction. In: *International Journal Universal Access in the Information Society* 1 (2002), Nr. 4, S. 235–236
- [MFC01] MÜLLER, Andreas ; FORBRIG, Peter ; CAP, Clemens: Model-based user interface design using markup concepts. In: JOHNSON, Chris (Hrsg.): *Interactive Systems: Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001*. Glasgow, Scotland, UK : Springer, June, 13-15 2001 (Lecture Notes in Computer Science 2220), S. 16–27

- [MHP00] MYERS, Brad ; HUDSON, Scott E. ; PAUSCH, Randy: Past, present, and future of user interface software tools. In: *ACM Trans. Comput.-Hum. Interact.* 7 (2000), Nr. 1, S. 3–28. – ISSN 1073–0516
- [Mic98] MICROSOFT INC.: The Windows Interface Guidelines for Software Design / Microsoft Inc. 1998. – Forschungsbericht
- [Mic03] MICROSOFT USER EXPERIENCE GROUP. *Aero User Experience Guidelines: Sampler for PDC 2003*. October 2003
- [Mic04] MICROSOFT CORPORATION: UI Guidelines / Microsoft Inc. Remond, WA, USA, 2004. – Guidelines
- [MM00] MARKOPOULOS, Panos ; MARIJNISSEN, Peter: UML as representation for interaction design. In: *Proceedings of Conference on Human-Computer Interaction Interfacing reality in the new millennium" (OZCHI 2000)*. Sydney, Australia, December, 4-8 2000, S. 240–249
- [MN90] MOLICH, Rolf ; NIELSEN, Jakob: Improving a human-computer dialogue. In: *Commun. ACM* 33 (1990), Nr. 3, S. 338–348. – ISSN 0001–0782
- [Mol04] MOLINA, Pedro J.: A Review to Model-Based User Interface Development Technology. In: *Making model-based UI design practical: usable and open methods and tools, workshop at the 9th International Conference on User Interfaces IUI'04*. Island of Madeira, Portugal, January 12-16 2004
- [Mor81] MORAN, Thomas P.: The Command Language Grammar, a representation for the user interface of interactive computer systems. In: *International Journal of Man-Machine Studies* 15 (1981), Nr. 1, S. 3–50
- [Mor83] MORAN, Thomas P.: Getting into a system: External-internal task mapping analysis. In: *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press, 1983. – ISBN 0–89791–121–0, S. 45–49
- [MPR03] MCCARRON, Shane P. ; PEMBERTON, Steven ; RAMAN, T. V.: XML Events: an events syntax for XML / World Wide Web Consortium (W3C). 2003 (<http://www.w3.org/TR/xml-events/>). – Recommendation
- [MPS02] MORI, Giulio ; PATERNO, Fabio ; SANTORO, Carmen: CTTE: Support for developing and analyzing task models for interactive system design. In: *IEEE Transactions on Software Engineering* 28 (2002), September, Nr. 9, S. 797–813
- [MPS04] MORI, Giulio ; PATERNO, Fabio ; SANTORO, Carmen: Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. In: *IEEE Transactions on Software Engineering* 30 (2004), January, Nr. 1, S. 507–520
- [MPWJ92] MARKOPOULUS, P. ; PYCOCK, J. ; WILSON, S. ; JOHNSON, P.: Adept – A task based design environment. In: *Proceedings of 25th Hawaii International Conference on System Sciences* IEEE, 1992, S. 587–596
- [MR86] MCCLELLAND, James L. ; RUMELHARDT, David E.: *Computational Models of Cognition*. Bd. 2: *Parallel distributed Processing*. 7. Cambridge, USA : MIT Press, 1986
- [MR01] MALKEWITZ, Rainer ; RICHTER, Kai: XML used for remote control of public kiosk systems (POI, POS). In: *Proceedings of E-Business and E-Work*, 2001
- [MR02a] METZKER, Eduard ; REITERER, Harald: Evidence-Based Usability Engineering. In: KOLSKI, Christophe (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces*. Valenciennes, France : Kluwer Academics, May, 15-17 2002, S. 323–336
- [MR02b] METZKER, Eduard ; REITERER, Harald: Use and Reuse of HCI Knowledge in the Software Development Lifecycle – Existing Approaches and What Developers Think. In: HAMMOND, J. (Hrsg.) ; GROSS, T. (Hrsg.) ; WESSON, J. (Hrsg.): *Usability: Gaining a Competitive Edge*. Dordrecht, NL : Kluwer Academics, 2002, S. 36–55

- [MS95] MAHAJAN, Rohit ; SHNEIDERMAN, Ben: A Family of User Interface Consistency Checking Tools. In: *Proceedings of the Twentieth Annual Software Engineering Workshop (SEL-95-004)*. Greenbelt, MD, USA, December 1995, S. 169–188
- [MS97] MAHAJAN, Rohit ; SHNEIDERMAN, Ben: Visual and Textual Consistency Checking Tools for Graphical User Interfaces. In: *IEEE Trans. Softw. Eng.* 23 (1997), Nr. 11, S. 722–735. – ISSN 0098–5589
- [MSLPGL05] MONTERO SIMARRO, Francisco ; LOZANO PÉREZ, María ; GONZALEZ LÓPEZ, Pascual: IDEALXML: an Experience-Based Environment for User Interface Design and pattern manipulation / Universidad de Castilla-La Mancha. Albacete, ES, 2005 (DIAB-05-01-4). – Technical Report
- [Mye92] MYERS, Brad A. (Hrsg.): *Languages for Developing User Interfaces*. London, UK : Bartlett Publishers, 1992
- [Mye95] MYERS, Brad A.: User interface software tools. In: *ACM Trans. Comput.-Hum. Interact.* 2 (1995), Nr. 1, S. 64–103. – ISSN 1073–0516
- [Mye01] MYERS, Brad A.: Using handhelds and PCs together. In: *Commun. ACM* 44 (2001), Nr. 11, S. 34–41. – ISSN 0001–0782
- [Mye04] MYERS, Brad A. *Mobile Devices for Control*. Presentation given at Fraunhofer IPSI, Germany. April 30 2004
- [NB01] NGO, David Chek L. ; BYRNE, John: Another look at a model for evaluating interface aesthetics. In: *International Journal of Applied Mathematic Computer Science* 11 (2001), Nr. 2, S. 515–535
- [ND86] NORMAN, Donald A. (Hrsg.) ; DRAPER, Stephen W. (Hrsg.): *User Centered System Design*. Hillsdale, New Jersey, USA : Lawrence Erlbaum Associates, 1986
- [New90] NEWELL, Allen: *Unified theories of cognition*. Cambridge, MA, USA : Harvard University Press, 1990. – ISBN 0–674–92099–6
- [New98] NEWMAN, William M.: A system for interactive graphical programming. In: *Seminal graphics: pioneering efforts that shaped the field*. New York, NY, USA : ACM Press, 1998. – ISBN 1–58113–052–X, S. 409–416
- [NFF+91] NECHES, Robert ; FIKES, Richard ; FININ, Timothy W. ; PATIL, Thomas R. Gruber and R. ; SENATOR, Ted E. ; SWARTOUT, William R.: Enabling technology for knowledge sharing. In: *AI Magazine* 12 (1991), December, Nr. 3, S. 36–56
- [Nie86] NIELSEN, Jacob: A virtual protocol for the computer-human interaction. In: *International Journal of Man-Machine Studies* 24 (1986), S. 301–312
- [Nie89] NIELSEN, Jakob: *Coordinating User Interfaces for Consistency*. Neuauflage 2002. San Francisco, CA, USA : Morgan Kaufmann Publishers, 1989 (The Morgan Kaufmann Series in Interactive Technologies)
- [Nie92] NIELSEN, Jacob: The Usability Engineering Lifecycle. In: *IEEE Computer* 25 (1992), March, Nr. 3, S. 12–22
- [Nie93] NIELSEN, Jakob: *Usability Engineering*. 12. San Francisco, CA, USA : Morgan Kaufmann, 1993
- [Nie03] NIELSEN, Jakob. *Usability 101: Fundamentals and Definition - What, Why, How (Jakob Nielsen's Alertbox)*. <http://www.useit.com/alertbox/20030825.html>. (retrieved: 27.01.2006) 2003
- [NLL95] NEWMAN, William M. ; LAMMING, Michael G. ; LAMMING, Mik: *Interactive System Design*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0201631628
- [NM03] NICHOLS, Jeffrey ; MYERS, Brad A.: Automatically Generating Interfaces for Multi-Device Environments. In: *Ubicomp 2003 Workshop: Multi-Device Interfaces for Ubiquitous Peripheral Interaction*, 2003
- [NM04] NICHOLS, Jeffrey ; MYERS, Brad A.: Report on the INCITS/V2 AIAP-URC Standard / Carnegie Mellon University. Pittsburgh, USA, 2004. – Forschungsbericht

- [NM05] NICHOLS, Jeffrey ; MYERS, Brad A.: Generating Consistent User Interfaces for Appliances. In: *Proceedings of 2nd Workshop on Multi-User and Ubiquitous User Interfaces(MU3I)*. San Diego, CA, USA, January 9 2005
- [NML04] NICHOLS, Jeffrey ; MYERS, Brad A. ; LITWACK, Kevin: Improving automatic interface generation with smart templates. In: *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*. New York, NY, USA : ACM Press, 2004. – ISBN 1–58113–815–6, S. 286–288
- [Nor83] NORMAN, Donald A.: Some Observations on mental models. In: GENTNER, D. (Hrsg.) ; STEVENS, A. L. (Hrsg.): *Mental Models*. Lawrence Erlbaum, 1983, S. 7–14
- [Nor88] NORMAN, Donald A.: *The Psychology of Everyday Things*. Basic Books, USA, 1988
- [Nov04] NOVAK, Gordon S. *Automatic Programming*. <http://www.cs.utexas.edu/users/novak-/autop.html>. (retrieved: 27.01.2006) 2004
- [NS72] NEWELL, A. ; SIMON, H. A.: *Human problem solving*. Englewood Cliffs, NJ, USA : Prentice Hall, 1972
- [NSA00] NGO, David Chek L. ; SAMSUDIN, Azman ; ABDULLAH, Rosni: Aesthetic measures for assessing graphic screens. In: *Journal of Information Science and Engineering* 13 (2000), S. 97–116
- [Oas05] OASIS OPEN. *OASIS (Organization for the Advancement of Structured Information Standards)*. <http://www.oasis-open.org>. (retrieved: 27.01.2006) 2005
- [OD83] OLSEN, Dan R. ; DEMPSEY, Elizabeth P.: SYNGRAPH: A graphical user interface generator. In: *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1983. – ISBN 0–89791–109–1, S. 43–50
- [OJNF00] OLSEN, Dan R. ; JEFFERIES, Sean ; NIELSEN, S. T. ; FREDRICKSON, William Moyesand P.: Cross-modal interaction using XWeb. In: *UIST'00: ACM SIGGRAPH Symposium on User Interface Software and Technology*, ACM, 2000, S. 191–200
- [Ols70] OLSON, Dan R.: *Cognitive Development: The child's acquisition of diagonality*. New York : Academic Press, 1970
- [OM95] OERTER, Rolf ; MONTADA, Leo: *Entwicklungspsychologie*. Weinheim, Germany : PVU Beltz, 1995
- [OR97] OPPERMAN, Reinhard ; REITERER, Harald: Software Evaluation using the 9241 evaluator. In: *Behaviour & Information* 16 (1997), Nr. 4/5, S. 232–245
- [Osg49] OSGOOD, Charles E.: The similarity paradox in human learning: a resolution. In: *Psychological Review* 56 (1949), S. 132–143
- [Ovi03] OVIATT, Sharon: Multimodal Interfaces. In: JACKO, Julie A. (Hrsg.) ; SEARS, Andrew (Hrsg.): *The Human-Computer Interaction Handbook*. Mahawah, New Jersey, USA : Lawrence Erlbaum Associates, 2003, S. 286–301
- [Pai86] PAIVIO, Allan: *Mental representations: A dual coding approach*. Oxford : Oxford University Press, 1986
- [Pat99] PATERNO, Fabio: *Model-based design and evaluation of interactive applications*. Berlin : Springer, 1999
- [Pay84] PAYNE, Stephen J.: Task Action Grammar. In: SHACKEL, B. (Hrsg.): *Proceedings of INTERACT'84*. Amsterdam, NL : North-Holland, 1984, S. 139–144
- [PBSK99] PATERNO, Fabo ; BREEDVELT-SCHOUTEN, Ilse M. ; DE KONING, Nicole: Deriving Presentations from Task Models. In: *Proceedings of the IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction*. Deventer, The Netherlands, The Netherlands : Kluwer, B.V., 1999. – ISBN 0–412–83520–7, S. 319–337

- [PCOM99] PUERTA, Angel ; CHENG, Eric ; OU, Tunhow ; MIN, Justin: MOBILE: user-centered interface building. In: *CHI '99: Proceedings of the SIGCHI conference on Human factors in computingsystems*. New York, NY, USA : ACM Press, 1999. – ISBN 0-201-48559-1, S. 426–433
- [PE98] PUERTA, Angel ; EISENSTEIN, Jacob: Interactively mapping task models to interfaces in MOBI-D. In: *In Proceedings of DSV-IS'98: Eurographics Workshop*, 1998
- [PE99] PUERTA, Angel ; EISENSTEIN, Jacob: Towards a general computational framework for model-based interface developmentsystems. In: *Knowledge-Based Systems* 12 (1999), S. 433–442
- [PE02] PUERTA, Angel ; EISENSTEIN, Jacob: XIML: a common representation for interaction data. In: *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*. New York, NY, USA : ACM Press, 2002. – ISBN 1-58113-459-2, S. 214–215
- [PE04] PUERTA, Angel ; EISENSTEIN, Jacob: XIML: A multiple user interface representation framework for industry. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-awareInterfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 119–148
- [Pei58] PEIRCE, Charles S.: *Collected Papers of Charles Sanders Peirce*. Bd. 1–8. Cambridge, MA : Harvard University Press, 1931–1958
- [Pfa85] PFAFF, G. E. (Hrsg.): *User Interface Management Systems*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1985. – ISBN 038713803X
- [Pfi02] PFISTERER, Christoph: *A Semantic Description Language for Platform-Independent Graphical Interfaces*, Institut für Visualisierung und Interaktive Systeme, Universität Stuttgart, Diploma Thesis, November 2002
- [PG86] PAYNE, Stephen J. ; GREEN, Thomas R. G.: Task-action grammars: A model of the mental representation of task languages. In: *Human-Computer Interaction* 2 (1986), S. 93–133
- [PG89] PAYNE, Stephen J. ; GREEN, Thomas R. G.: The structure of command languages: an experiment on task-action grammar. In: *International Journal on Man-Machine Studies* 30 (1989), S. 213–234
- [Pha00] PHANOURIOU, Constantinos: *UIML: A Device-Independent User Interface Markup Language*, Virginia Technical University, Diss., 2000
- [Pin00] PINHEIRO DA SILVA, Paulo: User Interface Declarative Models and Development Environments: A Survey. In: PALANQUE, Philippe (Hrsg.) ; PATERNÓ, Fabio (Hrsg.): *Interactive Systems: Design, Specification, and Verification (7th International Workshop DSV-IS Bd. 1946*. Berlin : Springer, June 2000, S. 207–226
- [Pix00] PIXLEY, Tom: Document Object Model (DOM) Level 2 Events Specification / World Wide Web Consortium. 2000 (<http://www.w3.org/TR/DOM-Level-2-Events/>). – Recommendation
- [PLV01] PRIBEANU, Costin ; LIMBOURG, Quentin ; VANDERDONCKT, Jean: Task Modeling for Context Sensitive User Interfaces. In: JOHNSON, Chris (Hrsg.): *Interactive Systems: Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001* Bd. 2220. Glasgow, Scotland, UK : Springer, June, 13-15 2001. – TY - CONF RP - IN FILE Y2 - 2001/06/13/, S. 49–68
- [PM99] PERLIN, Ken ; MEYER, Jon: Nested user interface components. In: *UIST '99: Proceedings of the 12th annual ACM symposium on User interfacesoftware and technology*. New York, NY, USA : ACM Press, 1999. – ISBN 1-58113-075-9, S. 11–18
- [PME86] POLSON, Peter G. ; MUNCHER, E. ; ENGELBECK, G.: A test of a common elements theory of transfer. In: *CHI '86: Proceedings of the SIGCHI conference on Human factors in computingsystems*. New York, NY, USA : ACM Press, 1986. – ISBN 0-89791-180-6, S. 78–83

- [PMM97] PATERNO, Fabio ; MANCINI, Cristiano ; MENICONI, Silvia: ConcurTaskTrees: A diagrammatic notation for specifying task models. In: HOWARD, Steve (Hrsg.) ; HAMMOND, Judy (Hrsg.) ; LINDGAARD, Gitte (Hrsg.): *Human-Computer Interaction, INTERACT '97, IFIP TC13 International Conference on Human-Computer Interaction* Bd. 96. Sydney, Australia : Chapman & Hall, July, 14-18 1997, S. 362–369
- [PR94] PALMER, S. ; ROCK, I.: Rethinking perceptual organization: The role of uniform connectedness. In: *Psychonomic Bulletin & Review* 1 (1994), S. 29–55
- [Pro02] PROSISE, Jeff: *Microsoft .NET – Entwicklerbuch*. Microsoft Press, August 2002
- [PS72] PASK, G. ; SCOTT, B. C. E.: Learning strategies and individual competence. In: *International Journal of Man-Machine Studies* 4 (1972), S. 217–253
- [Pue96] PUERTA, Angel: The Mecano project: comprehensive and integrated support for model-based interface development. In: *Computer-Aided Design for User Interfaces*. Namur, BE : Namur University Press, 1996, S. 19–36
- [Pue97] PUERTA, Angel: A model-based interface development environment. In: *IEEE Software* 14 (1997), S. 40–47
- [PV02] PRIBEANU, Costin ; VANDERDONCKT, Jean: Exploring Design Heuristics for User Interface Derivation from Task and Domain Models. In: KOLSKI, Christophe (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces*. Valenciennes, France : Kluwer, May, 15-17 2002, S. 103–110
- [Pyl73] PYLYSHYN, Zenon W.: What the mind's eye tells the mind's brain. In: *Psychological Bulletin* 80 (1973), S. 1–24
- [Pyl81] PYLYSHYN, Zenon W.: The imagery debate: Analogue media and tacit knowledge. In: *Psychological Review* 88 (1981), S. 16–45
- [Ras86] RASMUSSEN, Jens: *Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering*. New York : North Holland, 1986
- [Rau93] RAUTERBERG, Matthias: A product oriented approach to quantify usability attributes and the interactive quality of user interfaces. In: LUCZAK, H. (Hrsg.) ; CAKIR, A. (Hrsg.) ; CAKIR, G. (Hrsg.): *Work with display units 92*. Amsterdam, The Netherlands : North-Holland, 1993, S. 324–328
- [Rau95] RAUTERBERG, Matthias: Four different measures to quantify three usability attributes: 'feedback', 'interactive directness' and 'flexibility'. In: PALANQUE, P. (Hrsg.) ; BASTIDE, R. (Hrsg.): *Design Specification and Verification of Interactive Systems '95*. Berlin : Springer, 1995, S. 209–223
- [Rau96] RAUTERBERG, Matthias: How to measure and to quantify usability of user interfaces. In: ÖZOK, A. (Hrsg.) ; SALVENDY, G. (Hrsg.): *Advances in Applied Ergonomics*. West Lafayette, IN, USA : USA Publishing, 1996, S. 429–432
- [RBG00] ROMÁN, Manuel ; BECK, James ; GEFFLAUT, Alain: A Device-Independent Representation for Services. In: *Proceedings of Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'00)* IEEE, 2000, S. 73–81
- [RE03] RICHTER, Kai ; ENGE, Marita: Multi-modal framework to support users with special needs in interaction with public information systems. In: JACKO, Julie A. (Hrsg.): *Proceedings of Human-Computer Interaction with Mobile Devices and Services (Mobile HCI 2003)*. Berlin, Heidelberg, New York : Springer, September 8-11 2003, S. 286–301
- [Rea90] REASON, James: *Human Error*. Cambridge, UK : Cambridge University Press, 1990
- [Rei81] REISNER, Phyllis: Formal grammar and human factors design of an interactive graphics system. In: *IEEE Transactions on Software Engineering* 7 (1981), S. 229–240
- [Rei90] REISNER, Phyllis: What is consistency? In: DIAPER, Dan (Hrsg.) ; GILMORE, David J. (Hrsg.) ; COCKTON, Gilbert (Hrsg.) ; SHACKEL, Brian (Hrsg.): *Human-Computer Interaction, INTERACT '90, Proceedings of the IFIP TC13 Third Inter-*

- antional Conference on Human-Computer Interaction*. Cambridge, UK : North-Holland, August 1990, S. 175–181
- [Rei00] REITERER, Harald: Tools for Working with Guidelines in Different Interface Design Approaches. In: VANDERDONCKT, Jean (Hrsg.) ; FARENC, Christelle (Hrsg.): *Tools for Working with Guidelines, Proc. of the Int. Workshop on Tools for Working with Guidelines*. Espace Bellevue, Biarritz, France : Springer, October, 7-8 2000, S. 225–236
- [RH04a] RICHTER, Kai ; HELLENSCHMIDT, Michael: Interacting with the ambience: Multimodal interaction and ambient intelligence. In: *W3C Workshop on Multimodal Interaction*. Sophia-Antipolis, France : W3C, July 19-20 2004
- [RH04b] RICHTER, Kai ; HELLENSCHMIDT, Michael: Multimodal Mediators to the world as it is currently designed / Zentrum fuer Graphische Datenverarbeitung e.V. Darmstadt, Germany, 2004 (04i022-ZGDV). – Technical Report
- [RHE⁺82] ROACH, John ; HARTSON, H. R. ; EHRICH, Roger W. ; YUNTEN, Tamer ; JOHNSON, DeborahH.: DMS: A comprehensive system for managing human-computer dialogue. In: *Proceedings of the 1982 conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1982, S. 102–105
- [Ric01] RICHTER, Kai: Remote access to public kiosk systems. In: *Proceedings of the 1st International Conference on Universal Access in Human-Computer Interaction*, 2001
- [Ric02a] RICHTER, Kai: An advanced concept of assistance for mobile customers of POS. In: *Proceedings of E-Business and E-Works*, 2002
- [Ric02b] RICHTER, Kai: Generic interface descriptions using XML. In: *Proceedings of Computer Aided Design of User Interfaces (CADUI'02)*, 2002
- [Ric05a] RICHTER, Kai: A Transformation Strategy for Multi-Device Menus and Toolbars. In: *Proceedings of the Conference on Human Factors in Computing Systems (CHI2005)*. Portland, OR, USA, April 3-7 2005, S. 1741–1744
- [Ric05b] RICHTER, Kai: A Transformational Approach to Multi-Device Interfaces. In: *Proceedings of the Conference on Human Factors in Computing Systems (CHI2005)*. Portland, OR, USA, April 3-7 2005, S. 1126–1127
- [Ric06] RICHTER, Kai: Transformational Consistency. In: CALVARY, Gaelle (Hrsg.) ; PRI-BEANU, Costin (Hrsg.) ; SANTUCCI, Giuseppe (Hrsg.) ; VANDERDONCKT, Jean (Hrsg.): *Computer-Aided Design of User Interfaces VI, Proc. of 6th. Int. Conf. on Computer-Aided Design of User Interfaces CADUI 2006* Bd. VI. Bucharest, Romania : Springer-Verlag, Berlin, June 6.8 2006, S. 137–150
- [RJB97] RUMBAUGH, James ; JACOBSON, Ivar ; BOOCH, Grady: *Unified Modelling Language Reference Manual*. Addison-Wesley, 1997
- [RKM88] ROSSON, Mary B. ; KELLOGG, Wendy ; MAASS, Susanne: The designer as user: building requirements for design tools from designpractice. In: *Commun. ACM* 31 (1988), Nr. 11, S. 1288–1298. – ISSN 0001–0782
- [RNGS06a] RICHTER, Kai ; NICHOLS, Jeffrey ; GAJOS, Krzysztof ; SEFFAH, Ahmed: The Many Faces of Consistency in Cross-platform Design. In: *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*. Montreal, Canada, April, 22-23 2006
- [RNGS06b] RICHTER, Kai (Hrsg.) ; NICHOLS, Jeffrey (Hrsg.) ; GAJOS, Krzysztof (Hrsg.) ; SEFFAH, Ahmed (Hrsg.): *Proceedings of the CHI'06 Workshop on the Many Faces of Consistency in Cross-Platform Design*. Bd. 198. CEUR, August 28 2006 (CEUR-WS)
- [Roc73] ROCK, Irvin: *Orientation and form*. New York : Academic Press, 1973
- [Rog95] ROGERS, Everett M.: *Diffusion of Innovations*. 4th Edition. New York, USA : Free Press, 1995

- [Rog97] ROGERS, Everett M.: The Diffusion of Innovations Model and Outreach from the National Network of Libraries of Medicine to Native American Communities / National Network of Libraries of Medicine, Pacific Northwest Region, Seattle. 1997. – Available online at: <http://nnlm.gov/pnr/eval/rogers.html>
- [Ros04] VAN ROSSUM, Guido: *Python Reference Manual*. 2.3.1. Python Software Foundation. Inc., May 2004
- [Rot91] ROTE, Günther: Computing the minimum Hausdorff distance between two points set on a line under translation. In: *Information Processing Letters* 38 (1991), S. 123–127
- [Roy70] ROYCE, Winston W.: Managing the development of large software systems. In: *Proceedings of WESCON'70* IEEE, 1970. – (Proceedings of the 9th International Conference on Software Engineering, 1987, Monterey, CA., S. 328–338)
- [RRF04] ROTH, Volker ; RICHTER, Kai ; FREIDINGER, Rene: A PIN-entry method resilient against shoulder surfing. In: *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*. New York, NY, USA : ACM Press, 2004. – ISBN 1–58113–961–6, S. 236–245
- [Rub15] RUBIN, Edgar: *Visuell wahrgenommene Figuren*. Copenhagen, DK : Gyldenalske Boghandel, 1915
- [SA85] SINGLEY, Mark K. ; ANDERSON, John R.: The transfer of text-editing skill. In: *Journal of Man-Machine Studies* 22 (1985), S. 403–423
- [SA89] SINGLEY, Mark K. ; ANDERSON, John R.: *The Transfer of cognitive skill*. Cambridge, Massachusetts, USA : Harvard University Press, 1989 (Cognitive Science Series)
- [SAS98] STEPHANIDIS, Constantine ; AKOUMIANAKIS, Demosthenes ; SFYRAKIS, Alexandros: Universal accessibility in HCI: Process-oriented design guidelines and tool requirements. In: STEPHANIDIS, C. (Hrsg.) ; WARREN, A. (Hrsg.): *Proceedings of 4th ERCIM Workshop on User Interfaces for all*. Stockholm, Sweden, 1998
- [SBTZ04] SCHÜMMER, Till ; BORCHERS, Jan ; THOMAS, John ; ZDUN, Uwe: Human-Computer-Human Interaction Patterns. In: *Workshop on the human role in HCI Patterns CHI 2004*, 2004
- [SC02] SMYTH, Barry ; COTTER, Paul: Personalized adaptive navigation for mobile portals. In: VAN HARMELEN, Frank (Hrsg.): *In Proceedings of the 15th European Conference on Artificial Intelligence- Prestigious Applications of Artificial Intelligence (ECAI02)*. Lyon, France : IOS Press, Juli 2002, S. 608–612
- [Sch90] SCHVANEVELDT, Roger W. (Hrsg.): *Pathfinder associative networks: studies in knowledge organization*. Norwood, NJ, USA : Ablex Publishing Corp., 1990. – ISBN 0–89391–624–2
- [Sch96] SCHLUNGBAUM, Egbert: Model-based user interface software tools – current state of declarative models / Georgia institute of Technology. Georgia, USA, 1996 (96-30). – Technical Report
- [Sch98] SCHNEIDER, Hans-Jochen: *Lexikon der Informatik*. 4. Auflage. Munich, Germany : R. Oldenbourg Verlag, 1998
- [Sch05] SCHAFFER, E. *UI Design Newsletter – March, 2005: Seeking clarity on consistency*. <http://www.humanfactors.com/downloads/mar05.asp>. (retrieved: 27.01.2006) 2005
- [SCJS95] SHNEIDERMAN, Ben ; CHIMERA, Richard ; JOG, Ninad ; STIMART, David: Evaluating Spatial and Textual Style of Displays. In: *Proc. of Getting the Best from State of the Art Display Systems*. London, UK : The Society for Information Display, February 21–23 1995
- [Sea93] SEARS, Andrew: Layout Appropriateness: A metric for evaluating user interface widget layout. In: *IEEE Transactions on Software Engineering* 19 (1993), Nr. 7, S. 707–719

-
- [Sea95] SEARS, Andrew: AIDE: A step toward metric-based interface development tools. In: *Proceedings of the 8th ACM Symposium on User Interface Software and Technology (UIST)*. Pittsburgh, PA, USA : ACM, November 1995, S. 101–110
 - [Sed92] SEDGEWICK, Robert: *Algorithmen*. 2. Auflage. Reading, Mess., USA : Addison-Wesley, 1992
 - [Sha90] SHAN, Yen-Ping: MoDE: a UIMS for Smalltalk. In: *OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*. New York, NY, USA : ACM Press, 1990. – ISBN 0–201–52430–X, S. 258–268
 - [Sha91] SHACKEL, Brian: Usability — Context, Framework, Definition, Design and Evaluation. In: SHACKEL, Brian (Hrsg.) ; RICHARDSON, Simon (Hrsg.): *Human Factors for Informatics Usability*. Cambridge University Press, 1991
 - [She92] SHEPPARD, Sylvia: Report on the CHI'91 UIMS Tool Developers' Workshop. In: *SIGCHI Bull.* 24 (1992), Nr. 1, S. 28–31. – ISSN 0736–6906
 - [Shn98] SHNEIDERMAN, Ben: *Designing the user interface: strategies for effective human-computer-interaction*. 3. Addison-Wesley, Reading, Massachusetts, 1998
 - [Shn03] SHNEIDERMAN, Ben: *Leonardo's Laptop: Human needs and the new computing technologies*. Cambridge, Massachusetts : MIT Press, 2003
 - [SIKH82] SMITH, David C. ; IRBY, Charles ; KIMBALL, Ralph ; HARSLEM, Bill Verplanckand E.: Designing the Star User Interface. In: *Byte* 7 (1982), April, Nr. 4, S. 242–282
 - [SJ04a] SEFFAH, Ahmed ; JAVAHERY, Homa: *Multile User Interfaces: Cross-platform and context-aware interfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004
 - [SJ04b] SEFFAH, Ahmed ; JAVAHERY, Homa: Multiple User Interfaces: Cross-platform Applications and Context-awareInterfaces. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-awareInterfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 11–25
 - [SKD01] SEFFAH, Ahmed ; KECECI, N. ; DONYAEE, M.: QUIM: A framework for quantifying usability metrics in software qualitymodels. In: *Proceedings of the Second Asia-Pacific Conference on Quality Software (APAQS)*, 2001, S. 311–317
 - [SLN92] SZEKELY, Pedro A. ; LUO, Ping ; NECHES, Robert: Facilitating the exploration of interface design alternatives: the HUMANOIDmodel of interface design. In: *Proceedings of the Conference on Human Factors in Computing Systems*, 1992, S. 507–515
 - [SLN93] SZEKELY, Pedro A. ; LUO, Ping ; NECHES, Robert: Beyond interface builders: Model-based interface tools. In: ASHLUND, S. (Hrsg.): *Bridges between worlds. Proceedings on 1993 Conference on Human Factorsin Computing Science (Inter-CHI'93)*. New York, USA : ACM Press, April 1993, S. 383–390
 - [SLV⁺05] STANCIULESCU, Adrian ; LIMBOURG, Quentin ; VANDERDONCKT, Jean ; BENJAMINMICHOTTE ; MONTERO, Francisco: A transformational approach for multimodal web user interfaces based onUsiXML. In: *ICMI '05: Proceedings of the 7th international conference on Multimodalinterfaces*. New York, NY, USA : ACM Press, 2005. – ISBN 1–59593–028–0, S. 259–266
 - [SOBD93] SEIN, Maung K. ; OLFMAN, Lorne ; BOSTROM, Robert P. ; DAVIS, Sidney A.: Visualization ability as a predictor of user learning success. In: *Int. J. Man-Mach. Stud.* 39 (1993), Nr. 4, S. 599–620. – ISSN 0020–7373
 - [SP01] PINHEIRO DA SILVA, Paulo ; PATON, Norman W.: User Interface Modelling with UML. In: KANGASSALO, H. (Hrsg.) ; JAAKKOLA, H. (Hrsg.) ; KAWAGUCHI, E. (Hrsg.): *Proceedings of 10th European-Japanese Conference on Information Modelingand Knowledge Representation*. Amsterdam, The Netherlands : IOS Press, May 8–11 2001, S. 203–217
 - [SPZ04] STEPHANIDIS, Constantine ; PARAMYTHIS, Alexandros ; ZARIKAS, Anthony: The PALIO Framework for Adaptive Information Services. In: SEFFAH, Ahmed (Hrsg.)
-

- ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-awareInterfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 29–51
- [SS95] SAVIDIS, Anthony ; STEPHANIDIS, Constantine: Developing Dual Interfaces for Integrating Blind and Sighted Users: TheHOMER UIMS. In: *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, vol. 1, *Papers: Multimodal Interfaces* ACM, 1995, S. 106–113
- [SSC+95] SZEKELY, Pedro A. ; SUKAVIRIYA, Piyawadee N. ; CASTELLS, Pablo ; JEYAKUMARMUTHUKUMARASAMY ; SALCHER, Ewald: Declarative interface models for user interface construction tools: theMASTERMIND approach. In: BASS, Leonard J. (Hrsg.) ; UNGER, Claus (Hrsg.): *Engineering for Human-Computer Interaction. Proceedings of the IFIP TC2/WG2.7working conference on engineering for human-computer interaction*. Yellowstone Park, USA : Chapman & Hall, August 1995, S. 120 – 150
- [Sta05] STATISTISCHES BUNDESAMT: Internetnutzung in Deutschland / Statistisches Bundesamt. Wiesbaden, Deutschland, 14. März 2005. – Pressemitteilung
- [Ste99] STEPHANIDIS, Constantine: Designing for all in the Information Society: Challenges towards universalaccess in the information age / ERCIM ICST. 1999. – Research Report
- [Ste01] STEPHANIDIS, Constantine: *User Interfaces for all: Concepts, Methods, and Tools*. Mahawah, New Jersey, USA : Lawrence Erlbaum Associates, January 2001 (Human Factors and Ergonomics Series)
- [SV03] SOUCHON, Nathalie ; VANDERDONCKT, Jean: A Review of XML-Compliant User Interface Description Languages. In: JORGE, Joaquim A. (Hrsg.) ; NUNES, Nuno J. (Hrsg.) ; FALCÃO E CUNHA, João (Hrsg.): *Interactive Systems. Design, Specification, and Verification, 10th InternationalWorkshop, DSV-IS 2003* Bd. 2844. Berlin : Springer, June, 4-6 2003, S. 377–391
- [SW84] STREVELER, Dennis J. ; WASSERMAN, Anthony I.: Quantitative measures of the spatial properties of screen designs. In: SHACKEL, B. (Hrsg.): *Proceedings of the First International Conference on Human-Computer Interaction*. Amsterdam, The Netherlands : North-Holland, September 1984, S. 81–89
- [Sze96] SZEKELY, Pedro A.: Retrospective and Challenges for model-based interface development. In: *Computer-Aided Design of User Interfaces*. Namur, BE : Namur University Press, 1996, S. xxi–xliv
- [Tau88] TAUBER, M. J.: On mental models and the user interface. In: VAN DER VEER, G.C. (Hrsg.) ; GREEN, T. R. G. (Hrsg.) ; HOC, J.-M. (Hrsg.) ; MURRAY, D. (Hrsg.): *Working with computers: theory vs. outcome*. London, UK : Academic Press, 1988
- [TCC04] THEVENIN, David ; COUTAZ, Joelle ; CALVARY, Gaelle: A Reference Framework for the Development of Plastic User Interfaces. In: SEFFAH, Ahmed (Hrsg.) ; JAVAHERY, Homa (Hrsg.): *Multiple User Interfaces: Cross-platform Applications and Context-awareInterfaces*. West Sussex, England : John Wiley & Sons Ltd., 2004, S. 29–51
- [TDE03] TESCH, Joachim ; DIETZE, Leonhard ; ENCARNACÃO, L. M.: Personal Interfaces-To-Go: Mobile Devices for Data Exchange and Interactionin Heterogeneous Visualization Environments. In: *International Conference on Distributed Computing Systems Workshops (ICDCS)*. Providence, RI, USA : IEEE, May 2003, S. 290–293
- [TER99] TAENTZER, Gabi ; ERMEL, C. ; RUDOLF, Michael: The AGG approach: Language and tool environment. In: *Handbook of Graph Grammars and Computing by Graph Transformation, Volume2: Applications, Languages and Tools*. Singapore : World Scientific, 1999
- [The03] THE EUROPEAN COMMISSION. *European Year of people with disabilities*. 2003
- [Tho22] THORNDIKE, E. L.: The effect of changed data upon reasoning. In: *Journal of Experimental Psychology* 5 (1922), S. 33–38

- [TMP98] TAM, R. Chung-Man ; MAULSBY, David ; PUERTA, Angel R.: U-TEL: a tool for eliciting user task models from domain experts. In: *IUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces*. New York, NY, USA : ACM Press, 1998. – ISBN 0-89791-955-6, S. 77–80
- [TNB⁺95] TAYLOR, Richard N. ; NIES, Kari A. ; BOLCER, Gregory A. ; A.MACFARLANE, Craig ; ANDERSON, Kenneth M. ; JOHNSON, Gregory F.: Chiron-1: a software architecture for user interface development, maintenance, and run-time support. In: *ACM Trans. Comput.-Hum. Interact.* 2 (1995), Nr. 2, S. 105–144. – ISSN 1073-0516
- [Tog90] TOGNAZZINI, Bruce: Consistency. In: LAUREL, Brenda (Hrsg.): *The Art of Human-Computer Interface Design*. Reading, Massachusetts : Addison-Wesley, 1990, S. 75–79
- [Tog03] TOGNAZZINI, Bruce. *Ask Tog: First Principles of Interaction Design*. <http://www.asktog.com/basics/firstPrinciples.html>. (retrieved: 23.01.2006) 2003
- [Tra02] TRAETTEBERG, Hallvard: *Model-based User Interface Design*. Norway, Information Systems Group, Department of Computer and Information Sciences, Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology, Doctoral Thesis, May 2002
- [Tra04] TRAETTEBERG, Hallvard: Integrating dialog modelling and application development. In: *Making model-based UI design practical: usable and open methods and tools, Proceedings of the 9th International Conference on User Interfaces IUI'04*. Island of Madeira, Portugal, January 12-16 2004
- [Tra05] TRAPP, Marcus: The Influence of Unpredictability on Multiple User Interface Development. In: *Proceedings of 2nd Workshop on Multi-User and Ubiquitous User Interfaces (MU3I)*. San Diego, CA, USA, January 9 2005
- [Tul83] TULLIS, Thomas S.: The formatting of alphanumeric displays: A review and analysis. In: *Human Factors* 25 (1983), S. 657–682
- [Tul88a] TULLIS, Thomas S.: Screen Design. In: HELANDER, M. (Hrsg.): *Handbook of Human-Computer Interaction*. Amsterdam, NL : North Holland, 1988, S. 45–65
- [Tul88b] TULLIS, Thomas S.: A system for evaluating screen formats: Research and application. In: HARTSON, R. (Hrsg.) ; HIX, D. (Hrsg.): *Advances in Human-Computer Interaction*. Norwood, NJ, USA : Ablex Publishing, 1988, S. 214–286
- [Tve77] TVERSKY, Amos: Features of Similarity. In: *Psychological Review* 84 (1977), Nr. 4, S. 327–352
- [TW01] THORNDIKE, E. L. ; WOODWORTH, R. S.: The influence of improvement in one mental function upon the efficiency of other functions. In: *Psychological Review* 8 (1901), S. 247–261
- [Uni03] UNITED STATES GENERAL SERVICES ADMINISTRATION. *Section 508: The Road to Accessibility*. <http://www.section508.gov>. (retrieved: 27.01.2006) 2003
- [Van97] VANDERDONCKT, Jean: *Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive*. Namur, BE, Facultés Universitaires Notre-Dame de la Paix, PhD thesis, July 1997
- [Van04] VANDERHEIDEN, Gregg: Interaction for diverse users. In: SEARS, A. (Hrsg.): *The Human-Computer Interaction Handbook*. Mahawah, NJ, USA : Lawrence Erlbaum Associates, 2004, S. 397–400
- [Van05] VANDERDONCKT, Jean: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: PASTOR, O. (Hrsg.) ; FALCAO E CUNHA, J. (Hrsg.): *Proc. of 17th Conf. on Advanced Information Systems Engineering CAI-SE'05* Bd. 3520. Berlin, Germany : Springer-Verlag, 2005
- [VB93] VANDERDONCKT, Jean ; BODART, Françoise: Encapsulating knowledge for intelligent automatic interaction objects selection. In: *CHI '93: Proceedings of the*

- SIGCHI conference on Human factors in computingsystems*. New York, NY, USA : ACM Press, 1993. – ISBN 0-89791-575-5, S. 424–429
- [VFO01] VANDERDONCKT, Jean ; FLORINS, Murielle ; OGER, Frédéric: Model-based design of mobile user interfaces. In: DUNLOP, M.D. (Hrsg.) ; BREWSTER, S.A. (Hrsg.): *Proceedings of Mobile HCI'2001: Third International Workshop on Human Computer Interaction with Mobile Devices*. Lille, France, September 10 2001, S. 135–140
- [VG94] VANDERDONCKT, Jean ; GILLO, Xavier: Visual techniques for traditional and multimedia layouts. In: *AVI '94: Proceedings of the workshop on Advanced visual interfaces*. New York, NY, USA : ACM Press, 1994. – ISBN 0-89791-733-2, S. 95–104
- [VLF⁺01] VANDERDONCKT, Jean ; LIMBOURG, Quentin ; FLORINS, Murielle ; OGER, Frédéric ; MACQ, Benoît: Synchronised Model-Based Design of Multiple User Interfaces. In: SEFFAH, A. (Hrsg.) ; RADHAKRISHNAN, T. (Hrsg.) ; CANALS, G. (Hrsg.): *Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends HCI-IHM*. Lille, France, September, 10 2001
- [VLM⁺04] VANDERDONCKT, Jean ; LIMBOURG, Quentin ; MICHOTTE, Benjamin ; LAURENT-BOUILLON ; TREVISAN, Daniela ; FLORINS, Murielle: UsiXML: A user interface description language for specifying multimodal user interfaces. In: *W3C Workshop on Multimodal Interaction*. Sophia-Antipolis, France : W3C, July 19-20 2004
- [VP99] VANDERDONCKT, Jean ; PUERTA, Angel R.: Introduction to Computer-Aided Design of User Interfaces. In: *Proceedings of Conference on Computer Aided Design of User Interfaces (CADUI'99)*, 1999
- [VPM03] VAN DER VEER, Gerrit C. ; PUERTA MELGUIZO, Maria del C.: Mental Models. In: JACKO, Julie A. (Hrsg.) ; SEARS, Andrew (Hrsg.): *The human-computer interaction handbook: fundamentals, evolving technologies, and emerging applications*. Mahwah, NJ, USA : Lawrence Earlbaum Associates, 2003, Kapitel 3, S. 52–80
- [VS95] VIENNEAU, Robert L. ; SENN, Roy: A State of the Art Report: Software Design Methods / Air Force Research Laboratory - Information Directorate (AFRL/IF). Rome, NY, USA, March 1995 (F30602-89-C-0082). – Technical Report
- [VW00] VAN DER VEER, Gerrit ; VAN WELIE, Martijn: Task based groupware design: Putting theory into work. In: *Proceedings of DIS 2000*, 2000
- [Wah04] WAHLSTER, Wolfgang: ISTAG Report on Grand Challenges in the Evolution of the Information Society / European Commission, Office for Official Publications of the European Communities. Luxembourg, 2004. – Report
- [Wal05] WALDECK, Carsten: Liquid 2D scatter space for file system browsing: A visualization- and interaction technology optimized for MacOSX. In: *Proceedings of Infor Visualization (IV'2005)*. London, UK, July 6-8 2005
- [Wan93] WANDMACHER, Jens: *Mensch-Computer-Kommunikation: Grundwissen*. Bd. 2: *Software-Ergonomie*. Berlin, New York : De Gruyter, 1993
- [Wan03a] WANDKE, Hartmut: Assistance in human-machine interaction: a conceptual framework and a proposal for a taxonomy / Humboldt-Universität. 2003. – Forschungsbericht
- [Wan03b] WANDMACHER, Jens. *Mensch-Computer-Interaktion 2*. Vorlesungsunterlagen. Februar 2003
- [Wan05] WANDKE, Hartmut: Assistance in human-machine interaction: a conceptual framework and a proposal for a taxonomy. In: *Theoretical Issues in Ergonomics Science* 6 (2005), March-April, Nr. 2, S. 129–155
- [Was66] WASON, Peter C.: Reasoning. In: FOSS, B.M. (Hrsg.): *New horizons in psychology*. Hammondsworth : Penguin Books, 1966
- [WB90] WIECHA, Charles ; BOIES, Stephen: Generating user interfaces: Principles and use of ITS style rules. In: *Proceedings of ACM SIGGRAPH Symposium on User Interface Software and Technology*. New York, USA : ACM, October 1990, S. 21–30

- [WB04] WALDECK, Carsten ; BALFANZ, Dirk: Mobile Liquid 2D Scatter Space (ML2DSS). In: *8th International Conference on Information Visualisation (IV 2004)*. London, UK : IEEE Computer Society, July 14-16 2004, S. 494–498
- [WBBG89] WIECHA, Charles ; BENNETT, William ; BOIES, Stephen ; GOULD, John: Tools for generating consistent user interfaces. In: *Coordinating user interfaces for consistency*. San Diego, CA, USA : Academic Press Professional, Inc., 1989. – ISBN 0-12-518400-X, S. 107–130
- [WBBG90] WIECHA, Charles ; BENNETT, William ; BOIES, Stephen ; ANDSHARON GREENE, John G.: ITS: a tool for rapidly developing interactive applications. In: *ACM Trans. Inf. Syst.* 8 (1990), Nr. 3, S. 204–236. – ISSN 1046-8188
- [WBG88] WILSON, M. D. ; BARNARD, P. J. ; GREEN, T. R. G. ; MACLEAN, A.: Knowledge-Based Task Analysis for Human Computer System. In: VAN DER VEER, G. (Hrsg.) ; GREEN, T. R. (Hrsg.) ; HOC, J. (Hrsg.) ; MURRAY, D. M. (Hrsg.): *Working with computers: Theory vs. Outcome*. Academic Press, 1988
- [Wei91] WEISER, Mark: The Computer for the Twenty-First Century. In: *Scientific American* 265 (1991), September, Nr. 3, S. 94–104
- [Wei02] WEISS, Scott: *Handheld Usability*. West Sussex, England : John Wiley & Sons, Ltd., 2002
- [Wel01] VAN WELIE, Martijn: *Task-based user interface design*. De Boelelaan 1105, Amsterdam, The Netherlands, Vrije Universiteit Amsterdam, Doctoral Thesis, 17 April 2001
- [Wel05] VAN WELIE, Martijn. *Patterns in Interaction Design*. <http://www.welie.com>. (retrieved: 27.01.2006) 2005
- [Wer12] WERTHEIMER, Max: Experimentelle Studien über das Sehen von Bewegung. In: *Zeitschrift für Psychologie* 61 (1912), S. 161–265
- [Wer45] WERTHEIMER, Max: *Productive thinking*. New York, USA : Harper & Row, 1945
- [Wes02] WESSEL, Ivo: *GUI-Design: Richtlinien zur Gestaltung ergonomischer Windows-Applikationen*. 2. überarbeitete Auflage. München, Germany : Carl Hanser Verlag, 2002
- [WHB03] WALDECK, Carsten ; HESS, Daniel ; BALFANZ, Dirk: Mobile Liquid Information Spaces – Maximierung der Informationsdichte für visuelle Echtzeitsuche auf kleinen Screenflächen mit Hilfe von Transparenz und Wühlfunktionalität. In: SZWILLUS, Gerd (Hrsg.) ; ZIEGLER, Jürgen (Hrsg.): *Mensch & Computer 2003: Interaktion in Bewegung*. Stuttgart, Germany : Teubner, September 7-10 2003
- [Wil45] WILCOXON, F.: Individual comparisons by ranking methods. In: *Biometrics* 1 (1945), S. 80–83
- [Wil90] WILSON, David: *Programming with MacApp*. Reading, MA, USA : Addison-Wesley, 1990
- [WJ95] WILSON, Stephanie ; JOHNSON, Peter: Empowering users in a task-based approach to design experience and requirements. In: *Proceedings of Designing Interactive Systems (DIS'95)*. New York, USA : ACM Press, 1995, S. 25–31
- [WMM02] VAN WELIE, Martijn ; MULLET, Kevin ; MCINERNEY, Paul. *UI Patterns: A workshop for designers*. Workshop at CHI 2002. April 2002
- [WNPW02] WETZENSTEIN, Elke ; NITSCHKE, Julia ; POLKEHN, Knuth ; WANDKE, Hartmut: Assistenzsysteme für Unterhaltungselektronik im Haus und im PKW. In: *9. Dresdner Symposium für Psychologie der Arbeit*. Dresden, Germany, März 2002
- [Wor94] WORLD WIDE WEB CONSORTIUM (W3C): Web Accessibility Initiative (WAI) / World Wide Web Consortium (W3C). 1994. – Forschungsbericht
- [Wor01] WORLD WIDE WEB CONSORTIUM (W3C): Device Independence Activity / World Wide Web Consortium (W3C). 2001. – Forschungsbericht

- [Wor02] WORLD WIDE WEB CONSORTIUM (W3C): Multimodal Interaction Activity / World Wide Web Consortium (W3C). 2002. – Forschungsbericht
- [WV03] VAN WELIE, Martijn ; VAN DER VEER, Gerrit C.: Pattern Languages in Interaction Design: Structure and Organization. In: *Interact 2003*, 2003
- [WVE98] VAN WELIE, Martijn ; VAN DER VEER, Gerrit C. ; ELIENS, Anton: An ontology for task models. In: *Proceedings of 5th International Eurographics Workshop on Design Specification and Verification of Interactive Systems (DSV-IS'98)*, 1998, S. 55–70
- [WVE99] VAN WELIE, Martijn ; VAN DER VEER, Gerrit C. ; ELIENS, Anton: Breaking down Usability. In: *Proceedings of INTERACT'99*, 1999, S. 613–620
- [WVK00] VAN WELIE, Martijn ; VAN DER VEER, Gerrit V. ; KOSTER, Adrian: Integrated representations for task modeling. In: *Proceedings of 10th European Conference on Cognitive Ergonomics*, 2000, S. 129–138
- [X3.87] X3.113-1987, ANSI: PROGRAMMING LANGUAGES FULL BASIC / American National Standards Institute (ANSI). 1987. – American Standard
- [YJS03] YOUNG, Marc ; JOHNSON, Brian ; SKIBO, Craig: *Inside Microsoft Visual Studio .Net 2003*. Redmond, USA : Microsoft Press, 2003
- [ZAB06] ZIEFLE, Martina ; ARNING, K. ; BAY, Susanne: Cross platform consistency and cognitive compatibility: The importance of users' mental model for the interaction with mobile devices. In: *Proceedings of the CHI'2006 Workshop The many faces of consistency*. Montreal, Canada, April, 22-23 2006
- [ZB04] ZIEFLE, Martina ; BAY, Susanne: Mental Models of Cellular Phones Menu. Comparing older and younger users. In: BREWSTER, S. (Hrsg.) ; DUNLOP, M. (Hrsg.): *Mobile Human Computer Interaction*. Berlin : Springer, 2004, S. 25–37
- [Zim05] ZIMMERMANN, Gottfried. *Erweiterung des V2 Standards um semantische Modelle*. Persönliches Gespräch. September 2005
- [ZVG02] ZIMMERMANN, Gottfried ; VANDERHEIDEN, Gregg ; GILMAN, Al: Universal Remote Console - Prototyping for the Alternate Interface Access Standard. In: CARBONELL, N. (Hrsg.) ; STEPHANIDIS, C (Hrsg.): *Universal Access: Theoretical perspectives, practice and experience - 7th ERCIM UI4ALL Workshop*. Paris, France : Springer, October 2002 (Lecture Notes in Computer Science)

PERSÖNLICHE DATEN

Geboren am 19.07.1971 in Stuttgart-Bad Cannstatt
Familienstand: verheiratet

BERUFSERFAHRUNG

Wissenschaftlicher Mitarbeiter. ZGDV; Darmstadt — 04/01 - 07/06

Verantwortliche Mitarbeit in verschiedenen Forschungsprojekten, Koordination Mitarbeit bei der Erstellung von ca. zwölf Forschungsanträgen auf europäischer und nationaler Ebene, Technologieschulungen (XML, Web Services, etc.).

Ausgewählte Projekte:

- *EMBASSI (BMBF; 1997-2003)* — Leitung der Arbeitsgruppe Terminalsysteme. Konzeption und Implementierung einer Agentenplattform und einer multimodalen User Interface Darstellungsplattform.
- *kmed (BMBF; 2002)* — Projekt zur medienbasierten Lehre in der Medizin. Mitarbeit bei der Weiterbildung der Autoren.
- *Multimedia Werkstatt (2002-2003)* — Leitung und Verwaltung des Kursangebotes der Medienwerkstatt, sowie Schulung.
- *mKiosk (ZGDV; 2003)* — Konzeption und Umsetzung eines kontextsensitiven Dienstes zur mobilen Buchung und Planung von Reisen im ÖPNV.
- *SpacemantiX (EC; 2002-2005)* — Intelligente Einrichtungsplanung. Konzeption und Umsetzung eines Datenmodells und eines Werkzeuges zur Verknüpfung von 3D Modellen und Domänenkonzepten.
- *Reiseradar (Volkswagen AG, 2004-2005)* — Intelligentes kontextsensitives System zur Selektion von POI. Mitarbeit bei Konzeption, Umsetzung des Eigenschaftseditoren.
- *ADIVI (ZGDV; 2005)* — Konzeption und Umsetzung einer kollaborativen Hypervideo-Anwendung in Macromedia Flash.
- *ITB Mobil (ITB Berlin, 2005-2006)* — Mobiles Messenavigationssystem und individueller Planer. Mitarbeit an Konzeption, derzeit Umsetzung.
- *I2HOME (EC; 2006-2009)* — Unterstützung älterer und kognitiv eingeschränkter Nutzer im Umgang mit Haushaltsgeräten und Unterhaltungselektronik durch mobile Assistenten und Handlungsunterstützung.

User Interface Design Specialist. SAP AG; Walldorf — 08/06 - heute

Verantwortlich für Design und Konzeption von Entwicklungsumgebungen für mobilen Anwendungen sowie für das Design mobiler Anwendungen.

WEITERE ERFAHRUNG

Zivildienst, Kreiskrankenhaus; Waiblingen — 08/91 - 9/92

Medizintechnik. Testung und Wartung medizinischer Geräte nach Med-GV.

Praktikum. Daimler-Chrysler AG; Berlin — 02/93 - 03/93

Statistische Literatursauswertung. Untersuchung von Fahrerwahrnehmung anhand von Blickbewegungsmessungen und psychometrischer Daten.

Praktikum. Centre de la Recherche Scientifique (CNRS); Marseille — 07/97 - 08/97

Einarbeitung in die Programmierung kognitiver lokal-konnektionistischer Computermodelle zur Simulation der visuellen Worterkennung.

Studentische Hilfskraft. Philipps-Universität; Marburg — 09/97 - 10/99

Konzeption und Implementierung kognitiver Computermodelle zur Worterkennung und deren Anwendung in Simulationsstudien. Aufbau, Verwaltung und Pflege der psycholinguistischen Datenbank. Anleitung von Praktika und eines Tutoriums zu neuronalen Netzwerken.

Praktikum. Klinikum der Philipps-Universität; Marburg — 02/99 - 04/99

Klinisch-psychologische Diagnostik jugendlicher Patienten in der Klinik für Psychiatrie und Psychotherapie des Kinder- und Jugendalters. Durchführung zweier Studien zur Erforschung der Ursachen von Lese-Rechtschreib-Schwächen.

Studentische Hilfskraft mit Abschluss. Fraunhofer IGD; Darmstadt — 10/00 - 03/01

Mitarbeit in dem BMBF Forschungsprojekt EMBASSI.

AUSBILDUNG

Staufer-Gymnasium, Waiblingen (08/82 - 05/91) – Abitur; Note: 2,2 (Physik / Englisch)

Philipps-Universität, Marburg (03/93 - 04/00) – Diplom Psychologie; Note: «sehr gut»

KENNTNISSE

Fremdsprachen: Englisch (sehr gut in Wort und Schrift); Französisch (gut mündlich); Spanisch (mäßig in Wort und Schrift).

Methoden und Techniken: Usability Engineerig Methoden und User-Centric Development; Objektorientierte Entwicklung (RUP, UML); Sun Microsystems Java (J2SE); Microsoft .Net Framework und Compact Framework (C#); ECMA-Script; MATLAB; Web-Design (CSS, HTML, XHTML); Macromedia Flash (Actionscript); XML Technologien (DOM, XSD, XSLT, SVG, etc.); XML Web Services (SOAP, WSDL, UDDI).

Organisatorische Kenntnisse: Qualitätsmanagementbeauftragter; Erfahrung in Projektmanagement; Erfahrung in Konzeption und Durchführung von technischen Schulungen.

VERÖFFENTLICHUNGEN

- Kai Richter & Jeffrey Nichols (eds., in preparation). Cross-device Consistency. Information Systems Series. Springer-Verlag, Berlin.
- Kai Richter, Matthias Finke, & Dirk Balfanz (to appear). Hypervideo. In: Margherita Pagani (ed.): Encyclopedia of Multimedia Technology and Networking. Idea Group Inc, Hershey, PA, USA.
- Jan Alexandersson, Kai Richter, & Stephanie Becker (2006). I2HOME: Benutzerzentrierte Entwicklung einer offenen standardbasierten Smart Home Plattform. In Proceedings of USEWARE 2006 (Düsseldorf, 10-11 October 2006). Germany.
- Richter, K. (2006). Transformational Consistency, Chapter 11, in G. Calvary, C. Pribeanu, G. Santucci, J. Vanderdonckt (eds.), "Computer-Aided Design of User Interfaces V", Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Information Systems Series, Springer-Verlag, Berlin, 2006, pp. 137-150.
- Kai Richter, Jeffrey Nichols, Krzysztof Gajos, & Ahmed Seffah (2006). MAFOC '06: The Many Faces of Consistency in Cross-Platform Design. Proceedings of the CHI'06 Workshop on The Many Faces of Consistency in Cross Platform Design, Montreal, Canada, April 22-23, 2006. Published on CEUR-WS: Vol 198, 28-Aug-2006.
- Kai Richter, Jeffrey Nichols, Krzysztof Gajos, & Ahmed Seffah (2006). The Many Faces of Consistency in Cross-device User Interface Design. Workshop at CHI 2006 (22-23 April). Montreal, CN.
- Volker Roth & Kai Richter (2006). How to Fend off Shoulder Surfing. Journal of Banking and Finance, Issue on Frontiers in Payment and Settlement Systems. 30(6), 1727-1751.
- Kai Richter & Volker Roth (2005). Human-Computer Interaction and Security. In: Claude Ghaoui (eds.): Encyclopedia of HCI. Idea Group Inc, Hershey, PA, USA.
- Rainer Malkewitz & Kai Richter (2005). Semantic Product Properties and Shapes - What your Customer Already Knows. In: Cunningham, Paul and Cunningham, Miriam (eds.): Innovation and the Knowledge Economy, Part 1: Issues, Applications, Case Studies. Amsterdam; Berlin: IOS Press, 2005.
- Volker Roth, Tobias Straub, & Kai Richter (2005). Security and Usability Engineering with Particular Attention to Electronic Mail. International Journal of Human-Computer Studies. 63 (1-2), 51-73
- Kai Richter (2005). Doctoral Consortium - A Transformational Approach to Multi-Device Interfaces. Proceedings of the Conference on Human Factors in Computing Systems (CHI2005). Portland, OR, USA, pp. 1126-1127.
- Kai Richter (2005). A Transformation Strategy for Multi-Device Menus and Toolbars. Proceedings of the Conference on Human Factors in Computing Systems (CHI2005). Portland, OR, USA, pp. 1741-1744.
- Kai Richter & Michael Hellenschmidt (2004). Interacting with the Ambience: Multimodal Interaction and Ambient Intelligence. W3C Workshop on Multimodal Interaction. Sophia Antipolis, France: W3C
- Volker Roth, Kai Richter, & Rene Freidinger (2004). A PIN-entry method resilient against shoulder surfing. Proceedings of CCS'04. Washington D.C., USA: ACM.
- Kai Richter & Michael Hellenschmidt (2004). Multimodal Mediators to the Worlds as it is Currently Designed. Technical Report Number: 04i022-ZGDV, Zentrum für Graphische Datenverarbeitung e.V.
- Kai Richter & Marita Enge (2003). Multi-modal Framework to Support Users with Special Needs in Interaction with Public Information Systems. In: Luca Chittaro (eds.): Human-Computer Interaction with Mobile Devices and Services. Proceedings : Mobile HCI 2003. Berlin, Heidelberg, New York: Springer Verlag, pp. 286-301.
- Kai Richter (2002). Generic Interface Descriptions using XML. In: Kolski, Christophe (eds.): Computer-Aided Design of User Interfaces III. Proceedings 2002. Boston; Dordrecht; London: Kluwer Academic Publishers, pp. 275-282.
- Kai Richter (2002). An Advanced Concept of Assistance for Mobile Customers of POS. In: Brian Stanford-Smith (eds.): Information Society Technologies (IST): Challenges and Achievements in E-business and E-work. Proceedings Vol. 1 : e2002. Amsterdam; Berlin: IOS Press, pp. 266-272.
- Kai Richter (2001). Remote Access to Public Kiosk Systems. Proceedings of the Ninth International Conference on Human-Computer Interaction. New Orleans, LA, USA, pp. 195-199.
- Rainer Malkewitz & Kai Richter (2001). XML used for Remote Control of Public Kiosk Systems (POI, POS). In: Brian Stanford-Smith (eds.): Information Society Technologies (IST): E-work and E-commerce. Proceedings : Novel Solutions and practices for a Global Networked Economy.. Amsterdam, The Netherlands: IOS Press, pp. 178-183.
- Arnaud Rey, Arthur M. Jacobs, Florian Schmidt-Weigand & Kai Richter (2001). Evaluating computational models of reading with highly reliable data ($r > .95$). Annual Meeting of The European Society for Cognitive Psychology. Edinburgh, Scotland,
- Florian Schmidt-Weigand, H.-C. Nuerk, Kai Richter, Ralf Graf & Arthur M. Jacobs (1999). Ist der Nachbarschaftseffekt ein Effekt der Buchstabenverwechselbarkeit?. Proceedings of the 41. Tagung experimentell arbeitender Psychologen (TeaR). Leipzig, Germany

Kai Richter (1999). A functional unit model of visual word recognition. (Diploma Thesis) Philipps-Universität, Marburg.

Ralf Graf, H.-C. Nuerk, Arthur M. Jacobs, Johannes Ziegler & Kai Richter (1998). Factors of influence on Naming Latencies in the German language. Proceedings of the 40. Tagung experimentell arbeitender Psychologen (TeaP). Marburg, Germany,

BETREUTE STUDENTISCHE ARBEITEN

Oliver Burmeister (in Arbeit). Mobile User Interface Development based on Advanced Patterns. (Diplomarbeit). FH Saarbrücken.

Nina Valkanova (2006). Realisierung eines Hypervideosystems für mobile Endgeräte. (Diplomarbeit). Technische Universität Darmstadt.

Cristian Hofman (2006). Realisierung einer kollaborativen Hypervideoanwendung mit Macromedia Flash. (Diplomarbeit). Universität Siegen.

Darko Tolimir und Anja Wipfler (2006). Entwicklung von Interaktions- und Gestaltungskonzepten für eine kollaborative Hypervideoanwendung. (Diplomarbeit). Universität der Künste Berlin.

Frank Herold (2005). Entwicklung eines Application Programming Interface (API) für XForms Benutzerschnittstellen. (Diplomarbeit). Fachhochschule Wiesbaden.

Simone Lanz (2005). Realisierung einer Remote-Schnittstelle zwischen Anwendungslogik und User Interface Darstellung. (Diplomarbeit). Fachhochschule Mannheim.

Timo Kopp (2004). Entwicklung eines Systems zur plattformunabhängigen Spezifikation und Darstellung von User Interfaces. (Diplomarbeit). Fachhochschule Giessen-Friedberg.

Alexander Oros (2003). Framework zur zeitnahen Erfassung von Kompetenzen und Arbeitsschwerpunkten in Organisationen anhand des Mitarbeiter-Arbeitsverhaltens: Beispielhafte Implementierung basierend auf dem Surfverhalten. (Bachelor Arbeit). Fachhochschule Darmstadt.

WEITERE WISSENSCHAFTLICHE TÄTIGKEITEN

- CHI 2006: Organisation des Workshops "The Many Faces of Consistency"
- INTERACT 2005, CADUI 2006: Mitglied des Programmkomitee
- CHI 2005 - 2006, INTERACT 2005: Review von Beiträgen
- IBM Systems: Review von Beiträgen
- CHI 2005: Vorsitzender der Session "Mobile HCI"
- Review im INI-Graphicsnet Review System